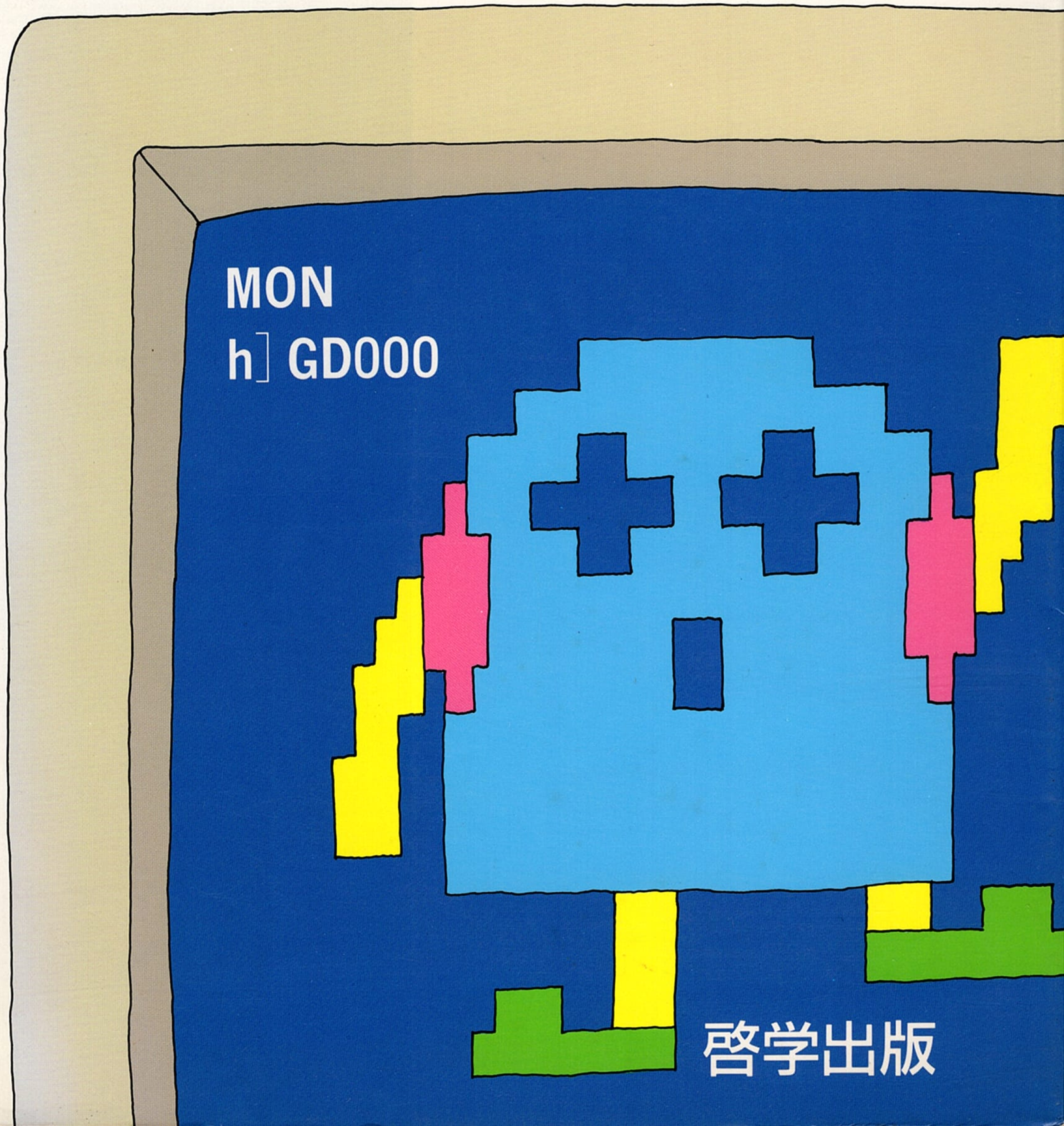
PC-8800シリーズ

はじめてのマシン語

対応機種══PC-8801mkⅡSR/TR/FR/MR/FH/MH

日高徹・著



啓学出版

PC-8800シリーズ はじめてのマシン語 日高徹・著 啓学出版

●著者に聞く

わたしのマシン語マスター法

日高　ぼくが，はじめてマシン語をマスターしようと思ったときは，手とり足とり教えてくれる人も，分かりやすい本もなかったから，とにかく他人の書いたソースリストを徹底的に解析したものです。
部分的にPC-88のモニタ・アセンブラで何をやってるのかを確かめながら，それをリストに書きこんでいきました。人間の言葉に置き換えていったわけですね。ジャンプすれば、行き先の番地をコピーして切り抜いて並べ換えたり，サブルーチンなんかは何回も呼ぶたびにコピーの切り抜きを重ねていくので，そこだけ張りつけたコピーで盛り上がったほどですよ。
こうすると「なるほど，このプログラムのこの部分は，こんなことをやっているのか」ということが納得いきます。いったん，こういった作業を経験しておくと，基本が分かるから，あとでずいぶん楽になりますよ。
わたしは，とにかく，やみくもにやりましたが，これから，はじめてマシン語をやってみようと思っている人たちに，この方法がいくらかでも役に立つかと考え，この本を書くことにしました。
というのも，前に『マシン語ゲームプログラミング』という本を出したときに，読者から「実際のプログラミングの説明に入る前に，もっと

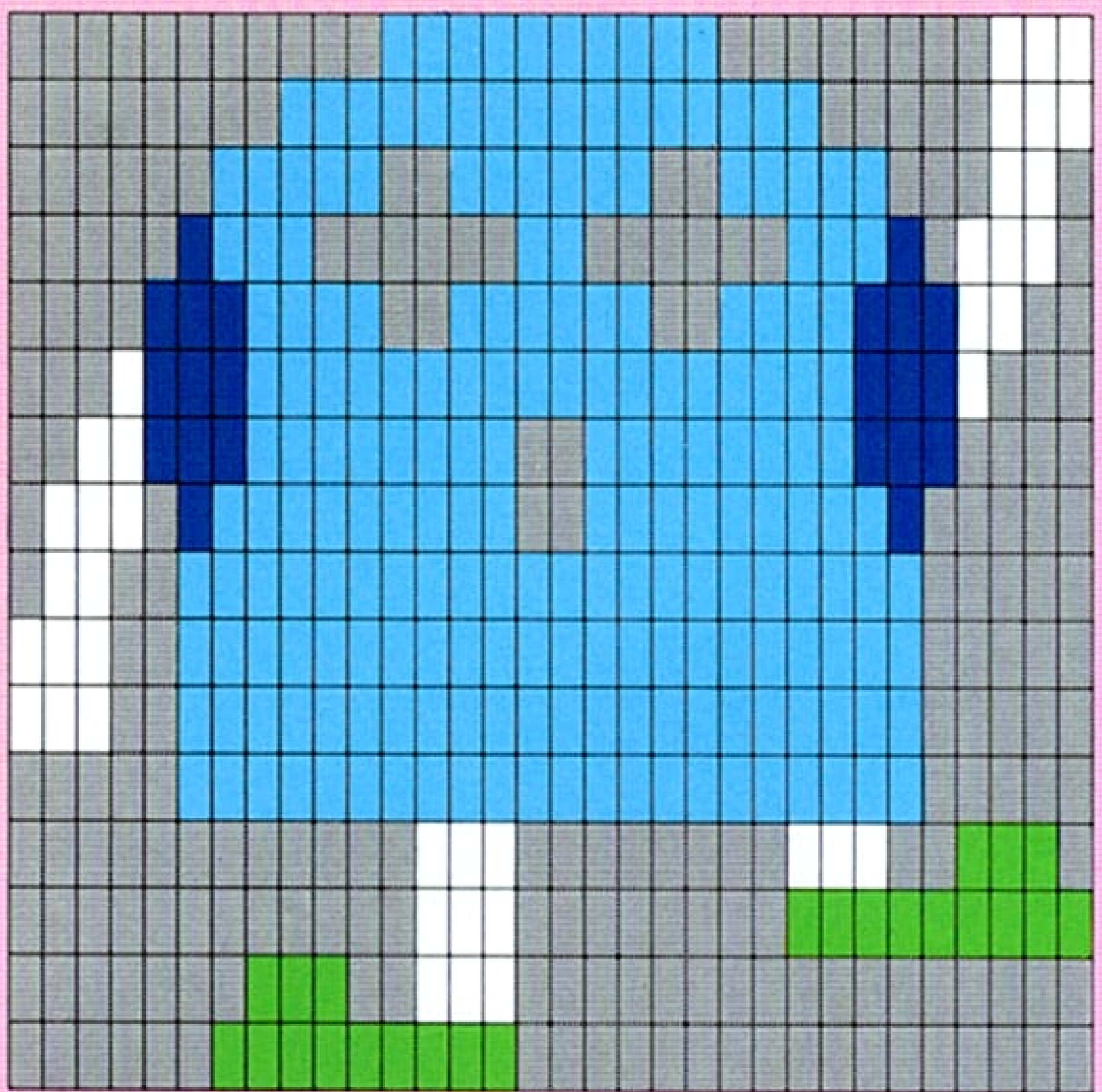
（後ろそでへ続く）

Designed by Yellow Bag

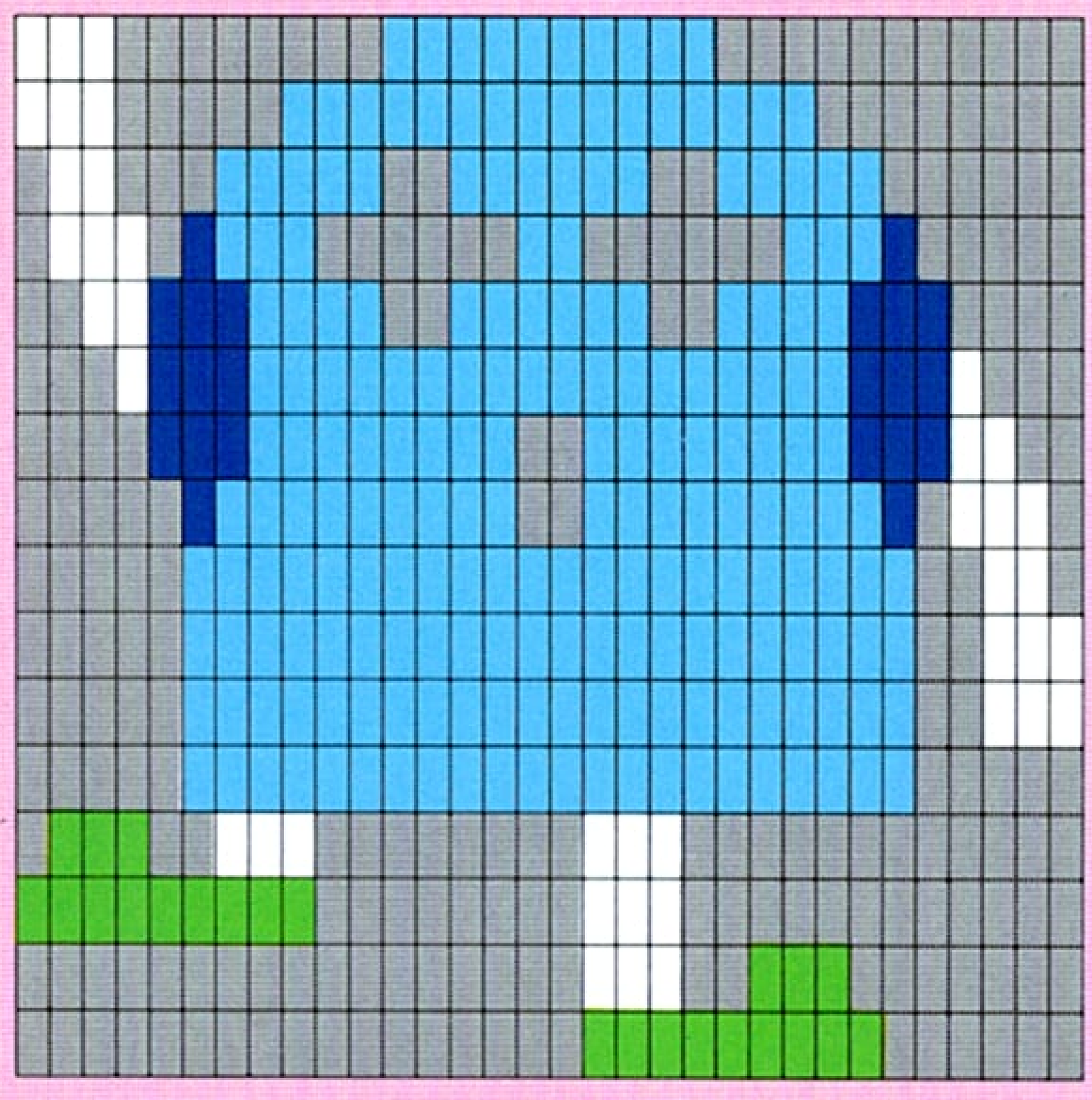
本書で作成するキャラクターパターン

カラーコード	0	1	2	3	4	5	6	7

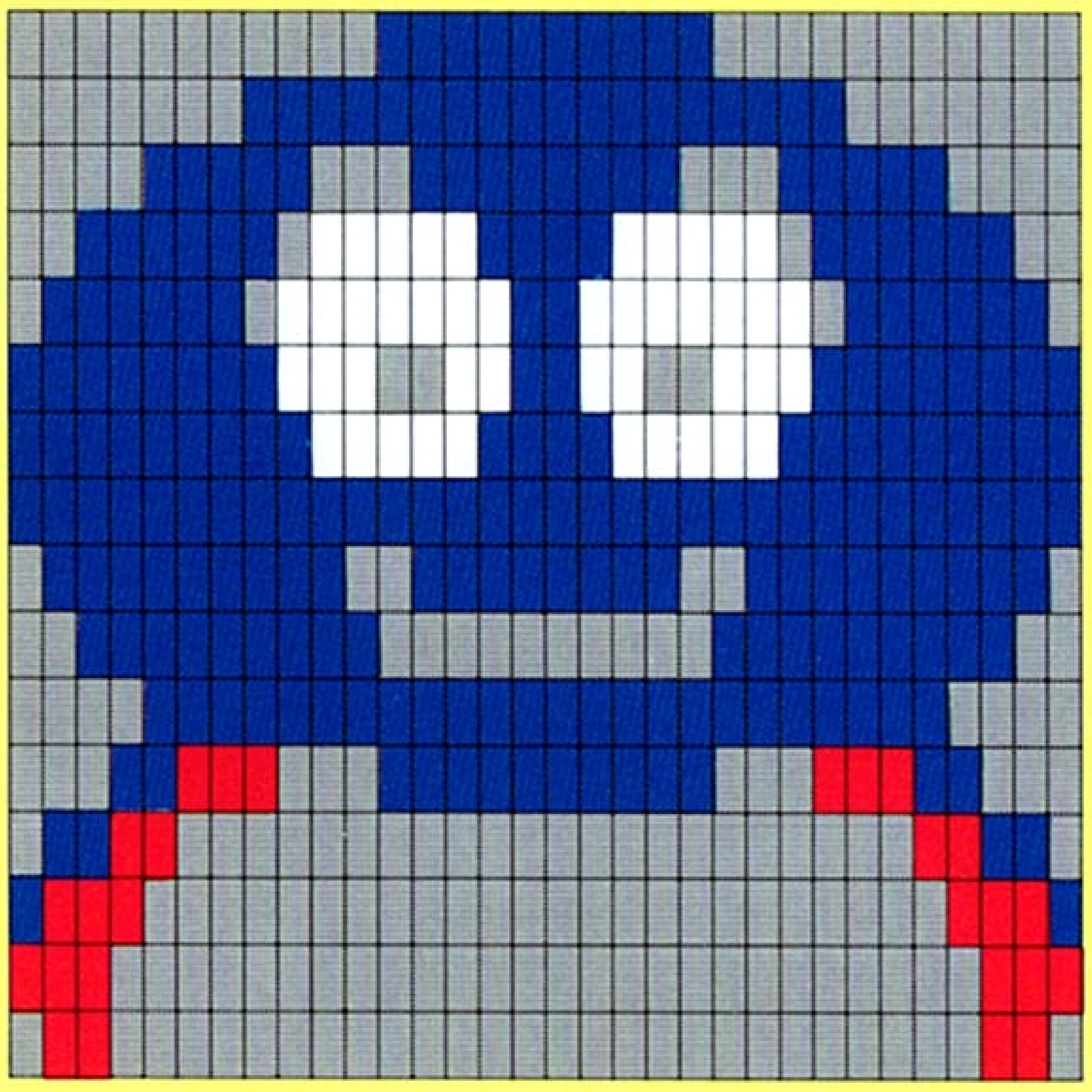
A1



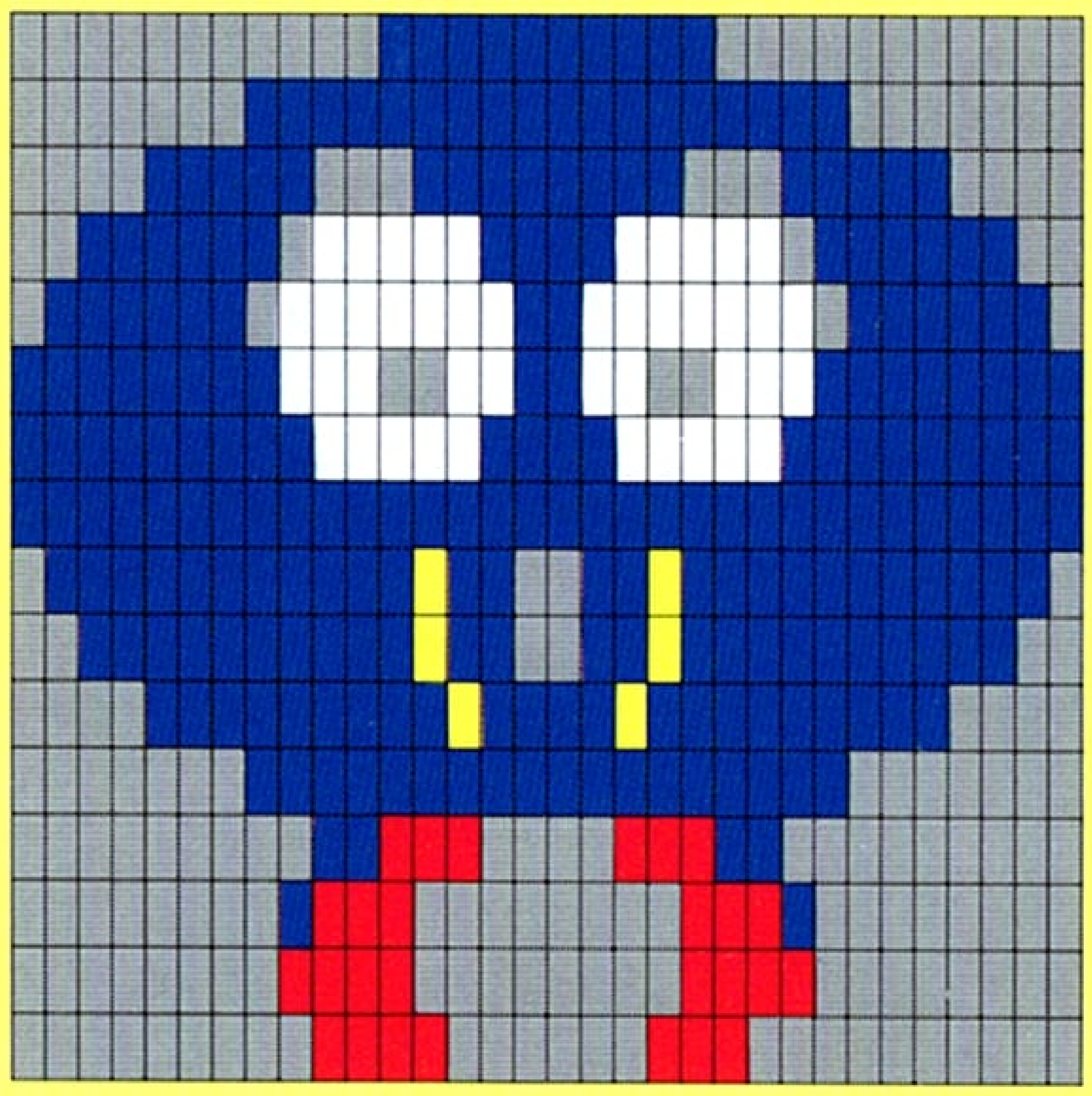
A2



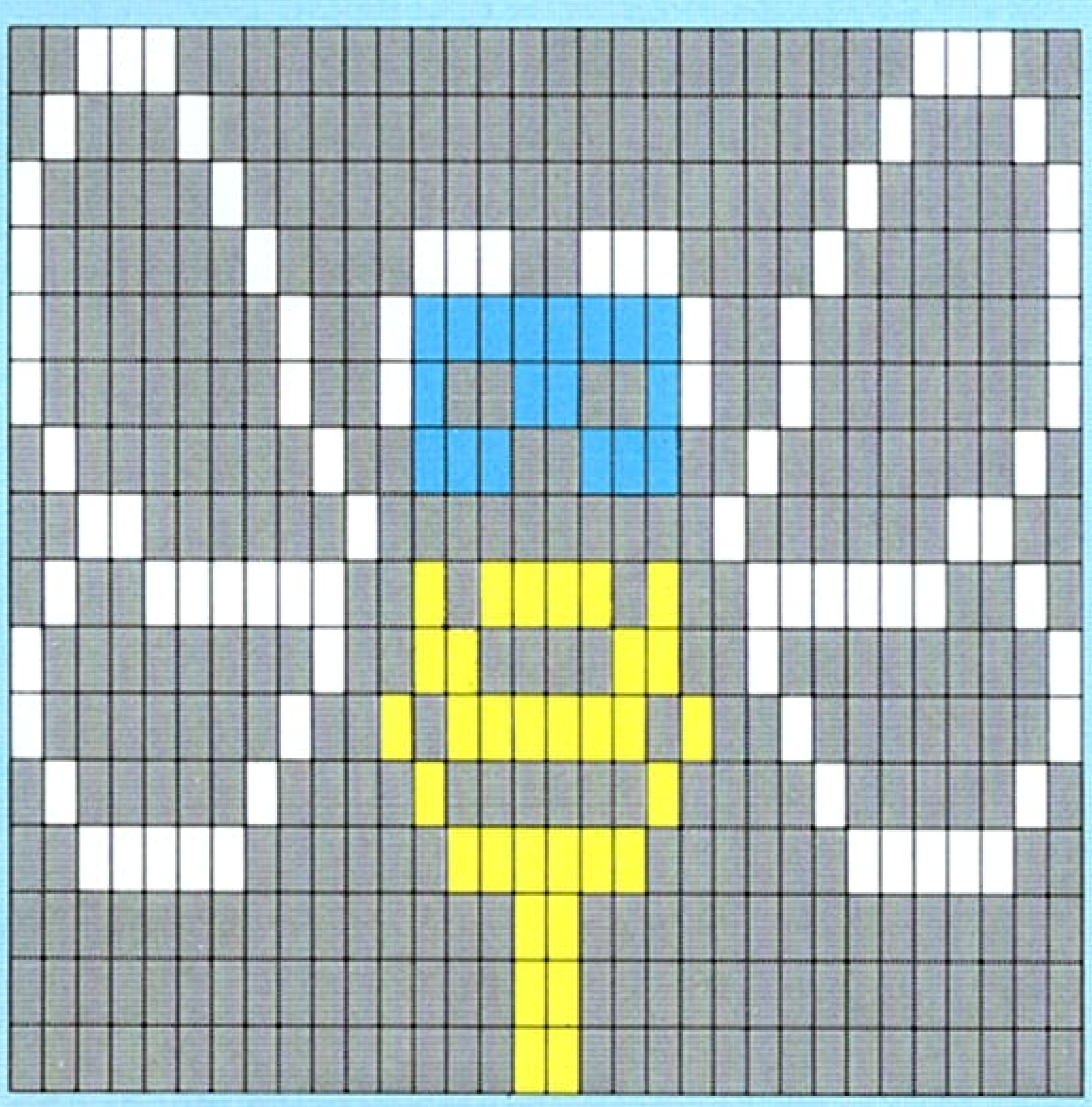
B1



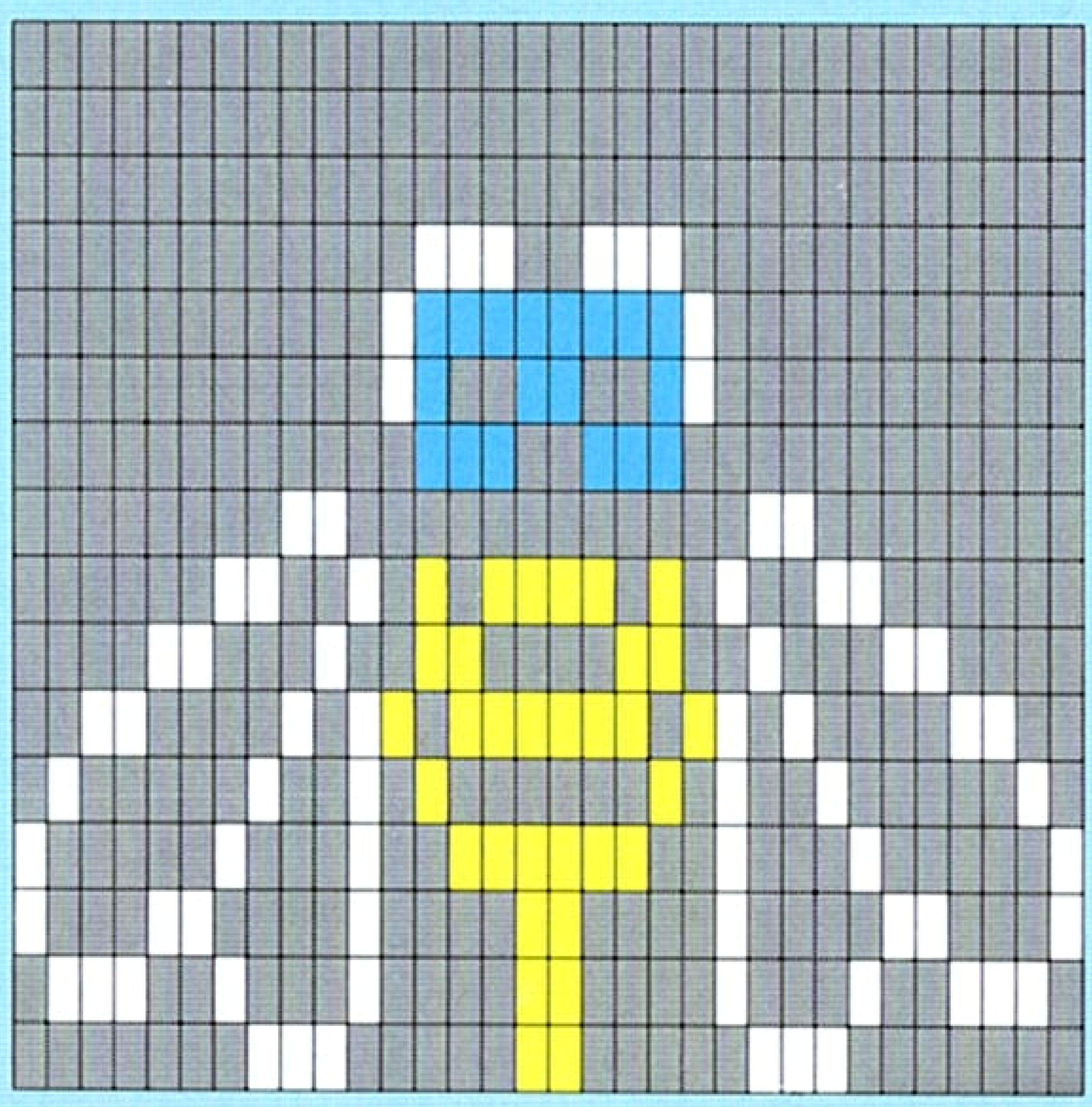
B2



C1



C2

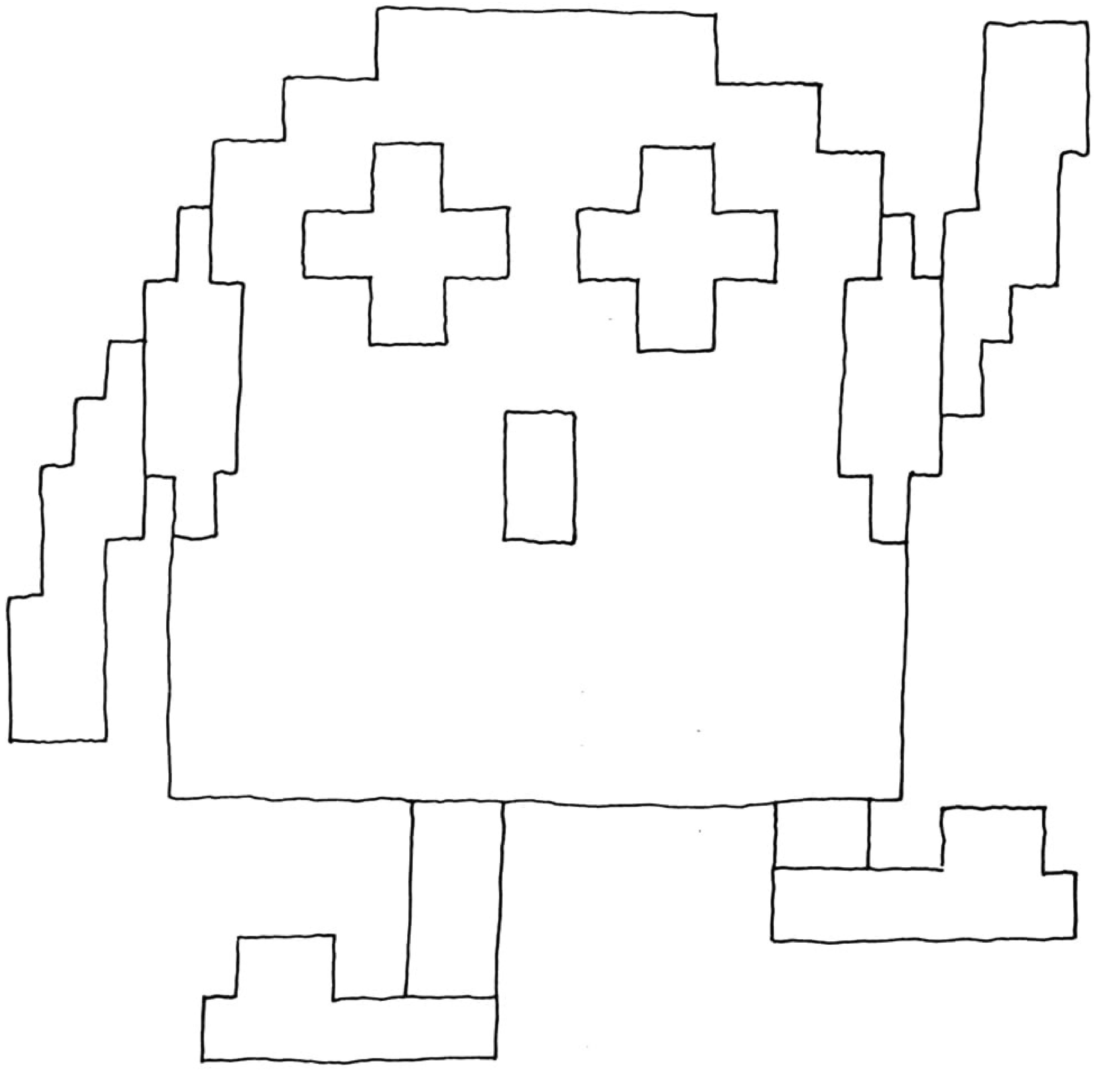


PC-8800シリーズ

はじめてのマシン語

対応機種══PC-8801mkⅡSR/TR/FR/MR/FH/MH

日高徹・著



啓学出版

はじめに

最近のパーソナルコンピュータの進歩は、まさに加速度的といってもいいくらい著しいものがあり、うれしい半面、限界を知らない恐しさを感じることがあります。

わが愛機、PC-8801も、最近の製品を見るとシリーズ初期の機種にくらべ、音楽やグラフィックスの面での機能が大きくアップしており、もはや8ビット機と16ビット機の差がないような感じになっています。それどころか、実際に、最新のPC-88VAでは、8ビット機の代名詞であった「88」が、なんと16ビットのコンピュータになってしまうという、突然変異的な進化をとげたほどです。

マシンの進歩につられてアップしているのが出版物の内容といえます。いつのまにやらレベルの高い記事を載せた雑誌や上級向けの書籍が増え、入門初級者向けのものと市場を二分する状況となりました。この結果、中級者は、無視されたような存在となっています。

パソコンユーザーで中級といえば、BASICをひととおり終え、マシン語の世界へ足を踏みいれようとしている大切な時期だと思うのですが、この間には深い川があって容易に渡れず、川岸をウロウロしている方がじつに多いように見受けられます。この障害がなくなれば、もっとたくさんの人々にマシン語を使ってもらうことができ、パソコン界全体のレベルアップにつながるはずなのに残念でなりません。

この解決のためには、マシン語をBASICのような気軽さで扱えるプログラミング環境（入力や実行方法）が必要です。それもできずにマシン語プログラミング云々といっても、話は先へ進まないのではないでしょうか。

わたしは、マシン語というのは言語ではなく、いわばアルファベットに相

当するものだ、と思っています。これをつづって１つの単語にし、さらには文章にするのがテクニックであり、アルゴリズムであると考えるものです。

本書は、マシン語国の対岸で橋を探して途方にくれているパソコンユーザーを対象に、PC-8801に内蔵されているモニタ・アセンブラを使って、マシン語を理解するうえでいちばん大切な基礎の基礎をわかりやすく解説しようと試みたものです。

この中で作る単語や文章は、親しみやすいようにゲームに素材を取っていますが、本当の目的は、ゲームを通じてBASICの世界から本格的マシン語の世界へと橋渡しをすることにあります。

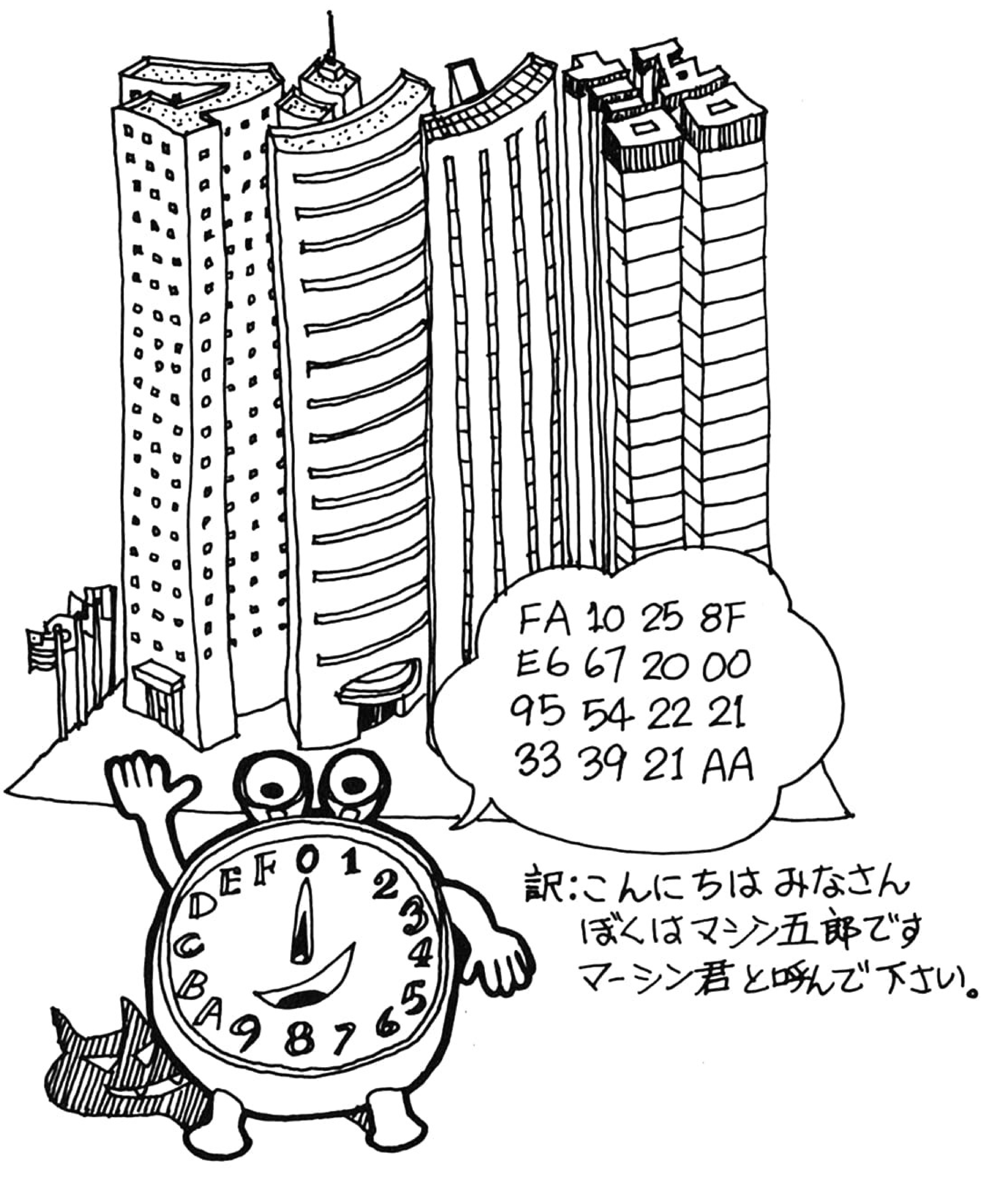
この橋は、わたし自身がはじめてマシン語に接し、マスターしようと努力した際に手探りで見つけた本当に小さな橋です。そのころのわたしは先輩のゲームデザイナーが書かれたソースリストを机上で解析し、モニタ・アセンブラで部分的に実行することを繰り返したあげく、この橋を架けることができました。まるで丸木を渡しただけかもしれませんが、その先は、確実に、雄大なマシン語の世界につながっていました。どうか一人でも多くの方がこの橋を渡ってもらえるようにと、願っています。

著　者

もくじ

第1章

BASIC とマシン語



マシン語というと、どんなイメージがありますか。

BASICを花にたとえるならば、マシン語は地面にもぐっている根っこのような、正体不明のものに思えるかもしれません。それゆえに「魔神語」などと、まるで化け物のように呼ばれることもあるくらいです。

本当に、そんなに難しいものなのでしょうか。花だって、根の部分を見ただけでは、何という花なのかわかりにくいものです。昔から、「陰と陽」「裏と表」ということばがあるように、コンピュータにも日の当たらない部分があるのです。そういう意味においては、マシン語はBASICなどの高級言語に比べると、あまり表には出られない悲しい運命を背負っているといえるかもしれません。

しかし、実際にはコンピュータはすべてマシン語で動いているのです。そこでこの章では、なにかにつけてマシン語と比較されるBASICについて、それが、いったいどんなものなのか、その実体はなんなのか、復習の意味もこめて軽くメスを入れてみることにしましょう。

1-1 BASIC の謎

一般に、コンピュータを手に入れて最初に接する言語はBASICというのが普通ですが、今あなたは、マシン語というものになにかしら興味を抱き、本書を読み出したわけです。それには、人それぞれ、さまざまな理由があると思いますが、やはり「BASICに不満や限界を感じたため」という方が圧倒的に多いのではないでしょうか。しかし、実はBASICプログラムもマシン語で動いていることに変わりはないのです。

■BASIC はマシン語だった

まずは、そのような現象を客観的に見つめられるように、BASICはマシン語であるということを納得してもらうことにしましょう。そのうえで、マシン語へとコマを進めることにします。

「BASICはBASIC……マシン語ではない!!」と、怒る方もいるかもしれません。けれどもコンピュータがマシン語しか理解できない以上、やはりBASICも最終的にはマシン語であることは、間違いのない事実なのです。いってみればBASICは、人間語と機械語の仲介をする通訳のようなものです。

たとえば、BASICのLINE文を使って画面上に線を引くと仮定します。

まずは命令文を入力してみましょう。

```
LINE (0, 0)-(100, 100), 7
```

見るからに、BASICということは明白なのですが、あなたがこの文をキーボードから入力して、画面に表示させたこと自体、すでにマシン語によって処理された結果なのです。さらにリターンキーを押し、ディスプレイ上に白い線が引かれた場合、その作業もすべてマシン語によってなされています。

つまり、BASICという人間にとってわかりやすい言語も、画面に表記されている文字（テキスト）だけがいわゆるBASICで、残るすべてはマシン語だったというわけです。

このように考えると、分厚いマニュアルが必要なほど多くの機能を持つBASICというのは、モノスゴイものだと感じられると思います。スピードが遅いのどうのとケチをつけられるような代物では、決してないのです。ただ、リアルタイム・ゲームを作るための言語ではありませんから、グラフィックに関する命令は十分とはいえない、というだけのことです。

■パズルのおもしろさ

さて、BASICに敬意を表したところで、実体はマシン語であるBASICを別の形のマシン語にするとなぜ不満が解消できるのか、考えてみましょう。

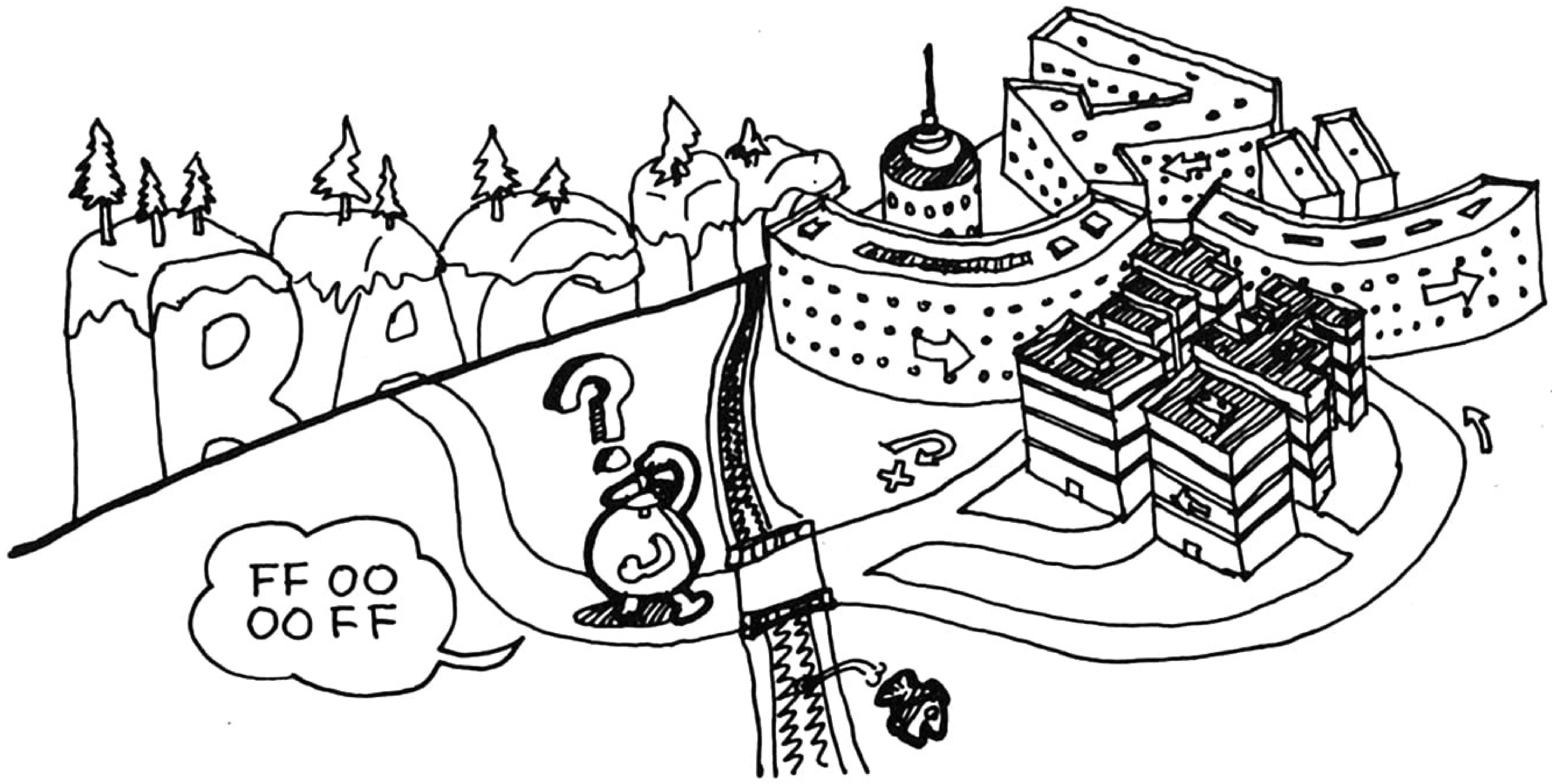
そもそもBASICにない機能（たとえばキャラクターパターンの高速表示など）については、新たに作るしかないのは当然のことですが、単にグラフィックルーチンのスピードアップを図りたい、などという場合には、自分の都合のいいように制限をつけてマシン語化すれば、目的が達成できることがあります。たとえば、グラフィック関係ではワールド座標やビューポート等を無視すれば、その分処理スピードは上がります。また、マシン語に慣れてくると、RAMの特徴を利用したプログラムを組むことも可能になり、やはりスピードアップにつながります。この辺については、少しずつ経験をつんでいくしかありません。

（注）RAMとはRandom Access Memoryの略で、読み書きが自由にできるメモリのことです。BASICプログラムを含めて、自分で作るプログラムはすべてこのRAM上に書かれていきます。これに対して、BASICプログラムを実行するためのマシン語は、最初からメモリに書かれており、読むことはできても書くことはできないようになっています。このようなメモリをROM（Read Only Memoryの略）といいます。

LINE文の例から、BASICはマシン語であるという事実、認めてくれましたか。要するに、コンピュータ自身はマシン語しか理解できない低級ロボットと思えばいいのです。低級ロボットの理解する言語ですから、マシン語と

いうのは１つひとつは大した内容ではありません。早くいえば、単純だということです。単純ということは簡単なわけですが、問題は、簡単すぎて１命令だけでは何もできないということです。

でも１命令でダメなら、小さな命令をいくつも組み合わせればいいはずですね。これは、まさしくジグソーパズルです。コンピュータ・プログラミングの原点は、パズルのおもしろさだというべきかもしれません。この本では、最後まで、キラ〜クな気持ちで、マシン語をパズルしてしまいましょう!!



1-2 マシン語の姿

マシン語の本なのに、いつまで読んでも肝心のマシン語にお目にかかれないのでは、イライラがつのる人が出てくるかもしれません。とくに、１つひとつの命令は簡単だといっておきながら、その正体を明かさないのでは、詐欺に等しいなどといわれかねません。本当は一応の予備知識がないと、やはりチンプンカンプンに違いないのですが、ここではマシン語の実体というものを、とりあえずのぞいてみることにします。ただし、全くわからないからといって、挫折したり、あきらめたりするのは早計ですので、念のため……。

■マシン語国へのパスポート

PC－8801でマシン語を扱うには、まず**モニタ**と呼ばれるマシン語操作モードに入らなければなりません。つまり、マシンを立ち上げた状態では自動的にBASICのモードになっているので、これを切り換えて、直接マシン語が使えるようにするのです。

これは、BASICの管理から離れて、完全にマシン語の世界に入ることですから、ふたたびBASICの世界にもどる方法も知っておかないと、あなたの手に負えなくなってしまうかもしれません。ちょうど、魔法を使うときには、その魔法の解き方を知らなければ、恐しいことになってしまうようなものです。

では、まずモニタ・モードへの入り方です。それにはMONという命令を使います。

```
MON⏎
h]▮ ←……カーソルがブリンクしている
```

画面の表示がこのようになると、モニタ・モードに入ったことを示します。

h] はPC-8801のマシン語のプロンプトです。これで、BASICの知識が全く通用しないマシン語の世界へ足をふみ入れたわけです。今は、この切り換え方がわかればいいので、とりあえず、またBASICモードへもどることにしましょう。そうするには、コントロールキーを押しながら英字のBのキーを押します。

```
h]^b  ←……CTRLキーを押しながら，Bキーを押す
OK
▮
```

記念すべきマシン語への第1歩は、あっけなく終わってしまいました。しかし、これにより、モニタとBASICのあいだの行き来は自由自在になったわけです。早速、もう一度MON ⏎ としてモニタ・モードへ入り、それからつぎのように入力してください。

```
h]DO⏎
```

■これがマシン語プログラム

入力したとたんに、画面にはなにやらわけのわからないものが――もちろん数字と英文字ということはハッキリとわかると思いますが、表示されたはずです。

```
h]DO
0000 F3 31 A0 E1 C3 E5 3B 00 7E E3 BE 23 E3 C2 93 03
h]▮
```

（注）これはSR/FR/MR/FH/MHでの内容です。PC-8801の旧モデルでは、0000のあとに表示されるF3以下の内容がすこし違っています。

このほかにも、画面の右側にゴチャゴチャしたものが見えていますが、これらはマシン語とは直接関係のないことですから無視して結構です。

それにしても、わけのわからない世界のようですね。なんだか、先が思いやられそうです。

今度は、つぎのように入力してください。

```
h]L0⏎
```

またまた、わけのわからない世界が現れてきました。

```
h]L0⏎
0000    F3          D1
0001    31 E1A0     LXI    SP, E1A0
0004    C3 3BE5     JMP    3BE5
0007    00          NOP
0008    7E          MOV    A, M
000A    BE          XTHL
000B    23          CMP    M
000C    E3          XTHL
000D    C2 0393     JNZ    0393
h]▮
```

D0 ⏎ 、L0 ⏎ のときといい、今度といい、どちらも、自信とヤル気をなくさせるようなイヤな雰囲気ですね。

結論だけをいうと、どちらもマシン語プログラムの一部であることは、想像の通りです。ただし、知らない世界というものは、わからなくて当たり前ですから、こんなことで難しそうだなんて思うことはありません。BASIC という地方から、いきなり地図も持たずにマシン語銀座へ入りこめば、だれだって戸惑うのは無理もないことです。

ここは、おぼえたての魔法で、速やかに BASIC へもどろうではありませんか。そして、もうすこし下準備をしてから、出直すことにしましょう。

1-3 コンピュータと二進数

われわれが、日常、何気なく使っていることでも、よく考えてみると必ずそこには一定のルール、あるいは前提が暗黙のうちに存在しています。

たとえば、八百屋さんの店先で80と表示してあれば、リンゴにしろミカンにしろ80円のことですが、中古車の展示場で80とあれば、これは80万円を意味しています。この例などは一般通念ともいえますが、そうはいってもこれが通用するのは、当然のことながら日本国内だけに限られるわけです。アメリカにはアメリカの、フランスにはフランスの貨幣単位があり、商習慣というものがあります。これを無視すると、その社会ではうまく生きていくことはできません。

あなたは、これからコンピュータというマシン語を使う国と、通訳(BASIC)を介さずに取引をしようというのです。これを商売と考えると、まずお金の数え方がわからなくては話になりません。しかし、それさえわかれば、辞書を片手に見振り手振りでなんとかなるのも商売ですから、ここは素直に相手に合わせる気持ちが大切です。ただ、コンピュータ国は数に対して非常にシビアですから、もしもあなたがこれまで「……ナァナァの感覚」で生きてきたならば、それだけは改めなければなりません。

■白か黒か

コンピュータが、なぜ数に対してシビアかというと、それは、すべての判断を、電気が通っているかいないか、すなわち完全白黒決着を基準として動いているからなのです。

この感覚を数字で表すならば、0か1かということになります。モノを数えるときも、0と1しか使えないのです。つまり、**二進数**の世界です。

われわれ人間は、ごく例外（時間や週など）を除いて、0～9までの数を

フルに用いる**十進数**の世界に慣れていますが、これは両手の指の合計が10本になるから都合がいいだけのことです。もしも指が左右８本ずつであったなら、人間社会も八進数、あるいは十六進数の世界になっていたに違いありません。

しかし、たとえそうであっても、これらは単に数値表現（記数法）の違いですから、絶対的な量に対してなんらの影響を与えるものではありません。つまり、十進数でリンゴ10個といえば、これは絶対的な量を示しており、何進数で表現しようが、リンゴそのものの量は不変というわけです。

なんだか、どこがいったいシビアなのかわからなくなってしまいましたが、要するにコンピュータは０と１しかない世界ですから、マシン語ということば自体も数値化されているということなのです。なんでもかんでも数値表現されるという点で、ひとつ間違えば意味不明となってしまう恐しい世界——ここがすなわち数に対してシビアというわけなのです。

これに対しては、人間側も便利なマシン語パーフェクト翻訳装置（？）を開発して対抗していますので、今から数字アレルギーになる必要はありません。これについては、いずれ明らかになりますのでお楽しみに……。

■８ビットのコンピュータとは

さて本題にもどりますが、この０と１で表される二進数の数値、その１桁１桁のことを**ビット**といいます。ですから、数値が１桁であれば１ビット、２桁であれば２ビットということになります。８桁であれば８ビット……。

アレッ、どこかで聞いたことがあることばですね。PC-8801は８ビットのコンピュータ、それも「究極の８ビットマシン」などといわれてきました。これは、なにを意味するのでしょうか。

８ビットの二進数とは、具体的には**図1.1**のように、０か１の数値が８つならんでいるだけのことです。それぞれの桁は、図にあるように、《ビット７》から《ビット０》という名前で区別ができるようになっています。そして、この８

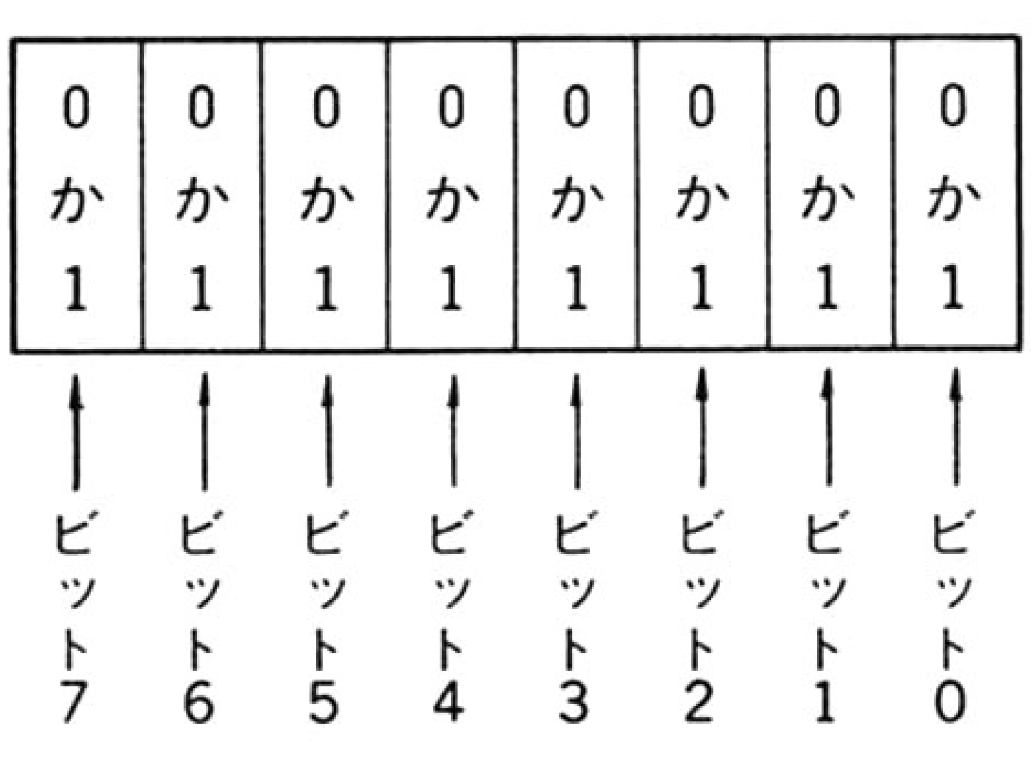


図1.1　８ビットの数値

ビットの二進数のまとまりを１単位として**１バイト**というのです。

８ビットのコンピュータというのは、一度にこの８桁（１バイト）の数値を扱うことができるコンピュータ、という意味なのです。

■８ビットで表せる数

では、８ビットの二進数で表現できる数値は、いったいいくつあるのでしょうか。わかりやすいように、十進数に換算しながらみてみましょう。

	最　小　値	最　大　値
８ビットの二進数	0000 0000	1111 1111
十進数	0	255

図1.2　８ビットで表現できる数値

十進数で表すと、たったこれだけです。「エッ、たったこれだけ!!　少ないナァ……」と感じれば、あなたはすでにマシン語を飲んでかかっているといえます。コンピュータといったって、しょせんはこんなものなのです。一度に1000＋1000の計算ができる人間のほうが、頭はイイに決まってますよね。

参考までに８桁の二進数を十進数に変換する式を載せておきますが、これはとくに重要なことではなく、電卓（二進数、十進数、十六進数の換算ができるもの）を利用するのが、なによりも安全確実な方法だと思います。

８桁の二進数＝

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0

$$十進数 = 2^7 \times B_7 + 2^6 \times B_6 + 2^5 \times B_5 + 2^4 \times B_4 + 2^3 \times B_3 + 2^2 \times B_2 + 2^1 \times B_1 + 2^0 \times B_0$$

図1.3　８桁の二進数を十進数に変換する方法

これまでの説明から、PC-8801は８桁の二進数、つまり８ビットを一度に処理できる能力を持っている、ということが明らかになりました。そして、これがマシン語の本当の姿である、ということをここで改めて覚えておいてください。表現できる範囲は、十進数でいうとたかだか０～255までのあいだです。思ったより小さな世界ではありませんか……。

1-4 二進数から十六進数へ

コンピュータ国の数字感覚、直取引ができそうな気がしてきたでしょうか。あまり自信に満ちた顔をしている人はいなさそうです。大体において、0～255までしか表現できないのに、8桁も必要だというのは面倒で仕方ありませんね。

十進法で8桁といえば、1000万の位です。家の値段にしろ、人口にしろ、一般にこれくらいの桁になれば、下4桁は0にまるめられてしまいます。23,456,789円なら、2345万円か、あるいは2300万円といったぐあいです。ところが、数字にシビアなコンピュータ国では、そのようなことは許されるはずがありません。

桁数が多くなった原因はじつに明白で、数値を0と1だけで表しているからです。それが証拠に、0～9までの数字を用いる十進数表記では、3桁(0～255)で表されています。しかし、この十進数というのも、手足の指を基準にして数えるときに便利というだけで、8桁の二進数を表すには、255という数字からもおわかりのように、決して区切りのいいものではありません。

■256を分解すれば

では、何進数で表すのがいちばんいいのでしょうか？

結論からいってしまえば、十六進数にするのがいちばんいいのです。つまり、0～255というのは256通りの数値表現ができるということですから、つぎのように分解していくことができるのです。

$$
\begin{aligned}
256 &= 16 \times 16 \\
&= (2 \times 2 \times 2 \times 2) \times (2 \times 2 \times 2 \times 2)
\end{aligned}
$$

ここにある2（＝2通り）というのは、ちょうど0と1（つまり2通りの

数）で表すことができる数字ですから、８ビットの数値を上位４ビットと下位４ビットに分けて、それぞれを十六進数表記すればいいということになります。

訳がわかったような、わからないような気分かもしれませんが、ここでは、あまり深く考えることはありません。ただ、８ビットの二進数を十六進数で表現するという事実だけを認識できれば、なにもいうことはありません。あとは自然に慣れてきます。

■十六進数の書き表し方

ところで、十六進数というのは、16まで数えたらつぎの桁にくり上がるという数え方ですから、15までは１桁で表現できるようになっていないと、数字でうまく書き表すことができませんね。ところが数字は０から９までの10通りしかありません。なにかうまい方法はないものでしょうか。そのため、１桁で10から15までを表すのにつぎのようにしています。

十 進 数：	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
十六進数：	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

つまり、10から15までを表すのに、アルファベットのA～Fを使っているわけです。こうすれば全部で16通りの書き分けができます。

では16はどう表現するのかというと、ここで桁が１つくり上がるわけですから、２桁で表記し10となります。１の位はくり上がったため０となり、上の位が１となるわけですね。

ところが、ここで、こまったことがおきました。たとえば17という数字は、十進数と十六進数では意味がちがうので、この区別をしなければ混乱してしまいます。そこで、十六進数に対しては数字のあとに $_H$ をつけて、十進数との区別ができるように記述することになっています。この $_H$ は Hexadecimal（ヘキサデシマル）、英語で「十六進の」という意味の頭文字を取ったものです。

というわけで十六進数の17は17_Hというぐあいに表記し「イチナナヘキサ」とか「イチナナ」と読みます。なお $_H$ がない場合は原則的に十進数です。念の

ため……。

この表現方法だけは、シッカリとおぼえなければなりません。参考のために、8ビットの世界で必要な256通りの数字について、十進数と十六進数と二進数を全部対比させておきましょう。15ページの表をみてください。

人間社会でふつう使っている十進数の世界では、0～9までの数字しかありませんでしたから、たとえばダースや時間など十進法ではない数え方が出てきても、十進数とまぜながらヤリクリしてきましたが、ここコンピュータ国ではこの十六進数が基本ですから、人間がそこをたずねて意志を伝えたいなら、どうしてもこの記数法を知っておく必要があります。そして、コンピュータ側もこの0～FFまでの十六進数を8桁の二進数と全く同じものとみなせるように設計されているのです。というより、8桁の二進数では、実際には、われわれは、コンピュータとコミュニケートできないのです。

モニタでは、十六進数、または八進数だけが使用できるようになっており、二進数は使用できません。また、八進数というものも実際には使用することはまずありませんので、結局のところ、この十六進数が、コンピュータ国の数字のすべてということになります。

そこで、問題となるのは二進数、あるいは十進数から十六進数への変換のしかたですが、人間には間違いがつきもですから、これは電卓にまかせるのが最もいい方法です。このような低レベルの作業で時間をかけたうえ、つまらぬミスでもしたら大変です。どうせなら、もっとマシン語の核心にふれる部分で苦労すべきです。

《例》

```
                上 位      下 位
                4ビット    4ビット
8 桁 の 二 進 数 = 1 1 1 1   0 0 0 1
                └──┬──┘   └──┬──┘
                   F          1

2桁の十六進数 = F1
```

			下位4ビット															
	二進数		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		十六進数	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
上位4ビット	0000	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0001	1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	0010	2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
	0011	3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
	0100	4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
	0101	5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
	0110	6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
	0111	7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
	1000	8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
	1001	9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
	1010	A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
	1011	B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
	1100	C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
	1101	D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
	1110	E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
	1111	F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

といっても、前述したような8ビットの二進数の場合は、上位4ビットと下位4ビットが2桁の十六進数の上位と下位にキチンと当てはまりますから、0～Fまでが数えられれば、わりと簡単に変換できるようになります。

では、ここまでのオサライをしましょう。

実際に扱われるマシン語は、2桁の十六進数である。

ここで学んだのは、このことだけです。簡単なものですね。

そこで、第2節（6ページ）でやったモニタでの作業、ページをもどしてもう一度やってみてください。まだまだ意味不明の点がありますが、マシン語の姿の一部、つまり2桁の十六進数ということが、こんどはハッキリと見えるようになったと思います。

1-5 メモリの構造

あなたがBASICでプログラムを組む場合、中身がなんであれ、行番号によってプログラムの流れを作り、RUN 命令によって実行するという工程は共通していました。

これがプログラムの基本であり、マシン語といえどもこのプロセスは同じものです。もっとも、マシン語には行番号やRUN 命令はありませんが、同じ機能をもった別のものがあると考えればいいだけのことです。

■マシン語を記憶させておく場所

マシン語とは、前節までの説明で、2桁の十六進数（00_H〜FF_H）であることがわかっていますが、これを実行までコンピュータに記憶させておく場所がメモリです。そして、1つひとつのメモリにはそれぞれアドレス（番地）がつけられており、これが行番号に相当します。このアドレスは、やはり十六進数で命名されています。

したがって、マシン語プログラムを実行させるときには、プログラムをスタートさせる場所（アドレス）を指定して走らせることになります。

では、走るのはだれでしょうか。つまり、メモリの中身を1つずつ読んで実行する選手のことですが、これはCPU（Central Processing Unit の略）というコンピュータの心臓部に当たるものが担当しています。

意外とアッサリ話を進めてしまいましたが、ここで重要なのは、マシン語のプログラミングとは、命令を直接メモリに書き込むことであり、そのメモリには区別のためのアドレスがついているという点です。

そのため、マシン語でプログラムを組むには、まずPC-8801のメモリ構成がどうなっているのかを知る必要があります。さもないと、プログラムは作れるようになったが、どのアドレスに置いたらいいのかわからない、という

ようなことになりかねません。

■メモリの大きさ

そこで、まずはメモリの総数ですが、これは理論上はいくらあってもかまいません。ただ、命令を読んで実行するCPUに、走れる限界があるのです。これをアクセス範囲といいます。8ビットのCPUの場合、一度に走れるメモリエリアは、最大で64Kバイト（1Kバイト＝1024バイト）に限られています。したがって理論上はともかく、実際には8ビットパソコンはこの範囲でしか仕事ができません。

64Kバイトとは、つぎの計算で求められるとおり、10000_Hバイトを意味します。

64Kバイト＝64×1024バイト
　　　　　＝65536バイト（十進数）
　　　　　＝10000_Hバイト（十六進数）

10000_Hバイトのメモリということは、アドレスでいうと0000_H番地から$FFFF_H$番地まで用意すれば足りることになります（0から数え始めると、$FFFF_H$で総メモリ数は10000_Hになりますね）。これが、8ビットCPUの走れる限界であるということを頭に入れたうえで、18ページのメモリ構造を見てください（図1.4）。これはPC-8801のメモリを図示したものです。

■PC-8801のメモリを解剖する

いきなり図1.4を見てしまうと、「なにがなんだか、サッパリわからん!!」となってしまいそうですが、この中のメインメモリと書かれた部分だけをまず見てください。図の左側に0000_Hから$FFFF_H$と書いてあるのがアドレスです。途中を省いてありますが、若い番地から順にならんでいます。

第1節で説明をしたBASICを動かすためのマシン語プログラムが、0000_H～$7FFF_H$番地をドンと占めているのがわかります。

その右どなりには、裏RAMと書かれたメモリが同じアドレスにありますが、ここはBASICプログラム本体が置かれるエリアです。

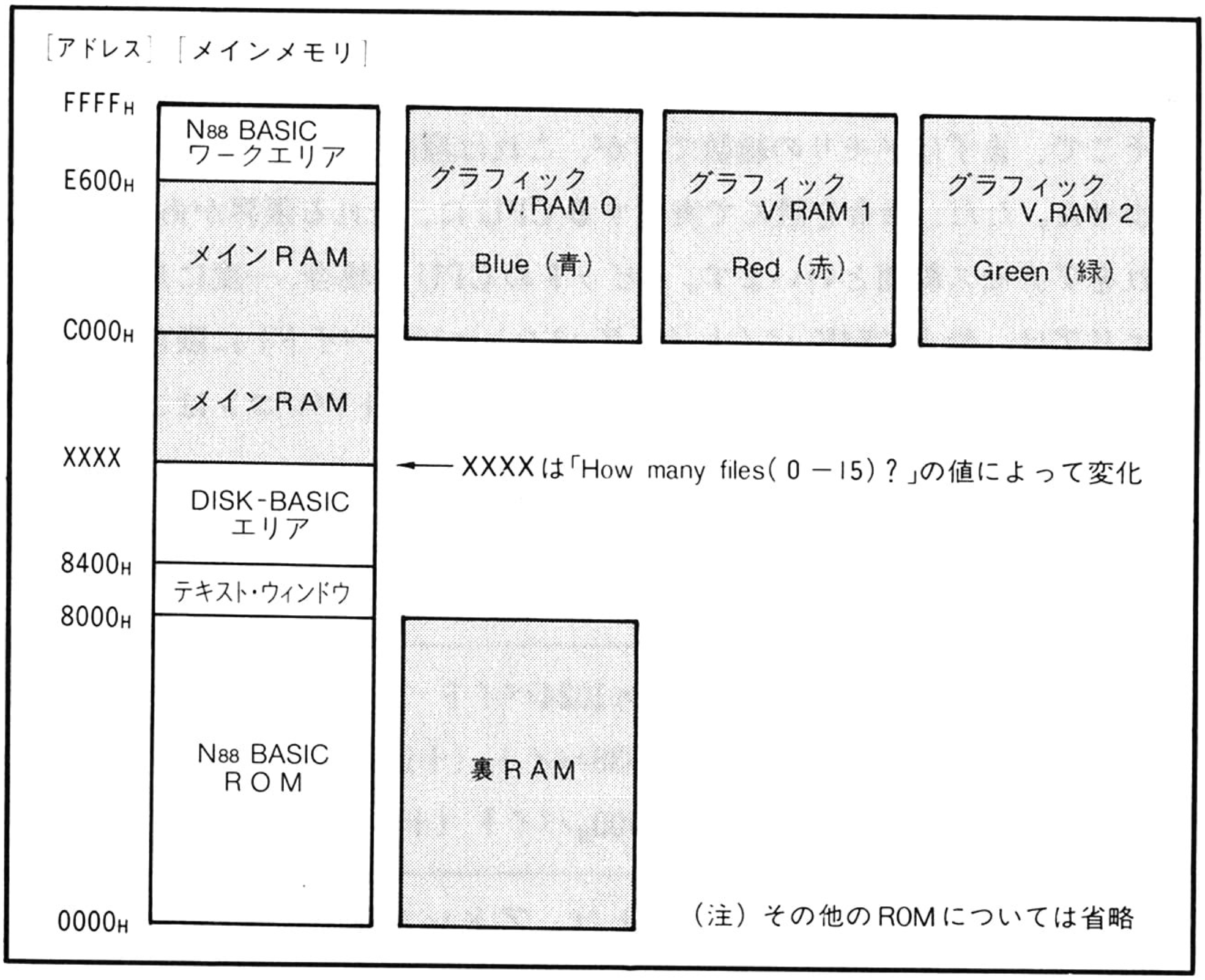


図1.4 PC-8801のメモリ構造

しかし、この場所はCPUが走ることができる範囲から離れてしまっているので、PC-8801ではテキストウィンドウというのぞき窓を作り、このBASICプログラムをメインメモリから読めるようにしています。これが、8000_H～$83FF_H$番地（400_Hバイト分）までの部分です。この窓を通せば、裏RAMにあるBASICプログラムのうち、400_Hバイトの範囲なら好きな部分を見ることができます。

DISK-BASICエリアとは、DISK-BASIC用のマシン語プログラムが入る部分です。

そしてメインRAMが、ユーザーがプログラムを置けるエリアとなります。

N_{88}-BASICワークエリアとは、おもにBASICを動かすマシン語プログラ

ムが、好きなように使うエリアと思えばいいでしょう。

$C000_H$～$FFFF_H$番地には、メインメモリの横に3種類のグラフィックV.RAMがありますが、一度にアクセスできる範囲は64Kバイトに限られていますから、同時に使うことはできません。これらはバンク切り換えという方法で使用できるようになっています。

グラフィックV.RAMの内容についてはここでは触れませんが、バンク切り換えとは、スイッチにより、どれか1つをメインメモリと直接結びつけることで、8ビットのCPUで実質64K以上のメモリを管理できるようにすることです。

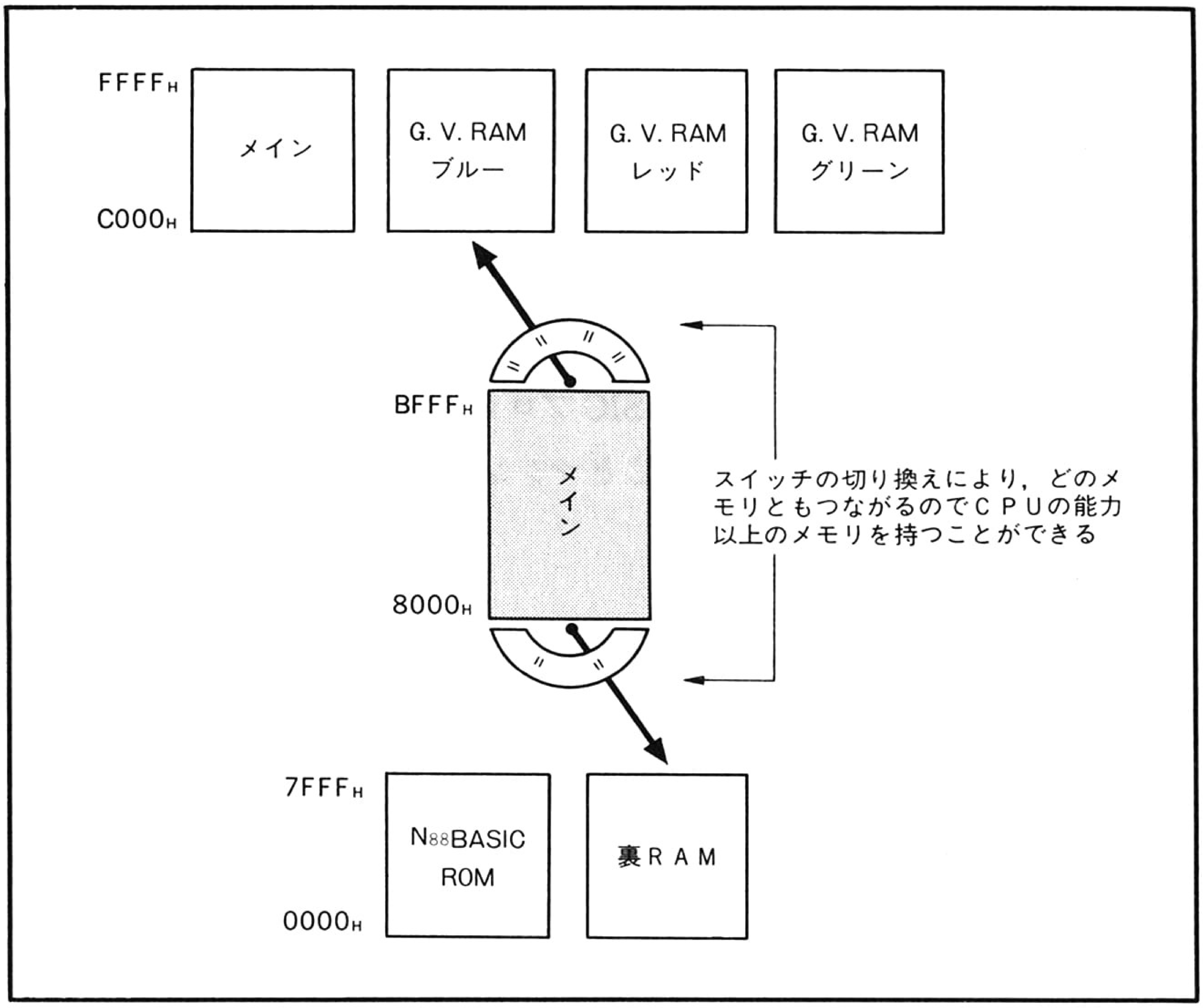


図1.5　バンク切り換え

裏RAMも、テキストウィンドウから読み出すばかりでなく、バンク切り換えによる使用も可能となっています。

■使用エリアの宣言

さて、メインRAMがユーザーの使用できるエリアであると書きましたが、このエリアはBASICの変数などにも使用されるところです。そこで下手をすると、おたがいに相手のメモリ内容を破壊しあうことにもなりかねません。そのためマシン語でプログラムを組む際には、問題が起きないよう使用エリアの宣言をする必要があるのです。これはBASICのCLEAR文により、つぎのように行います。

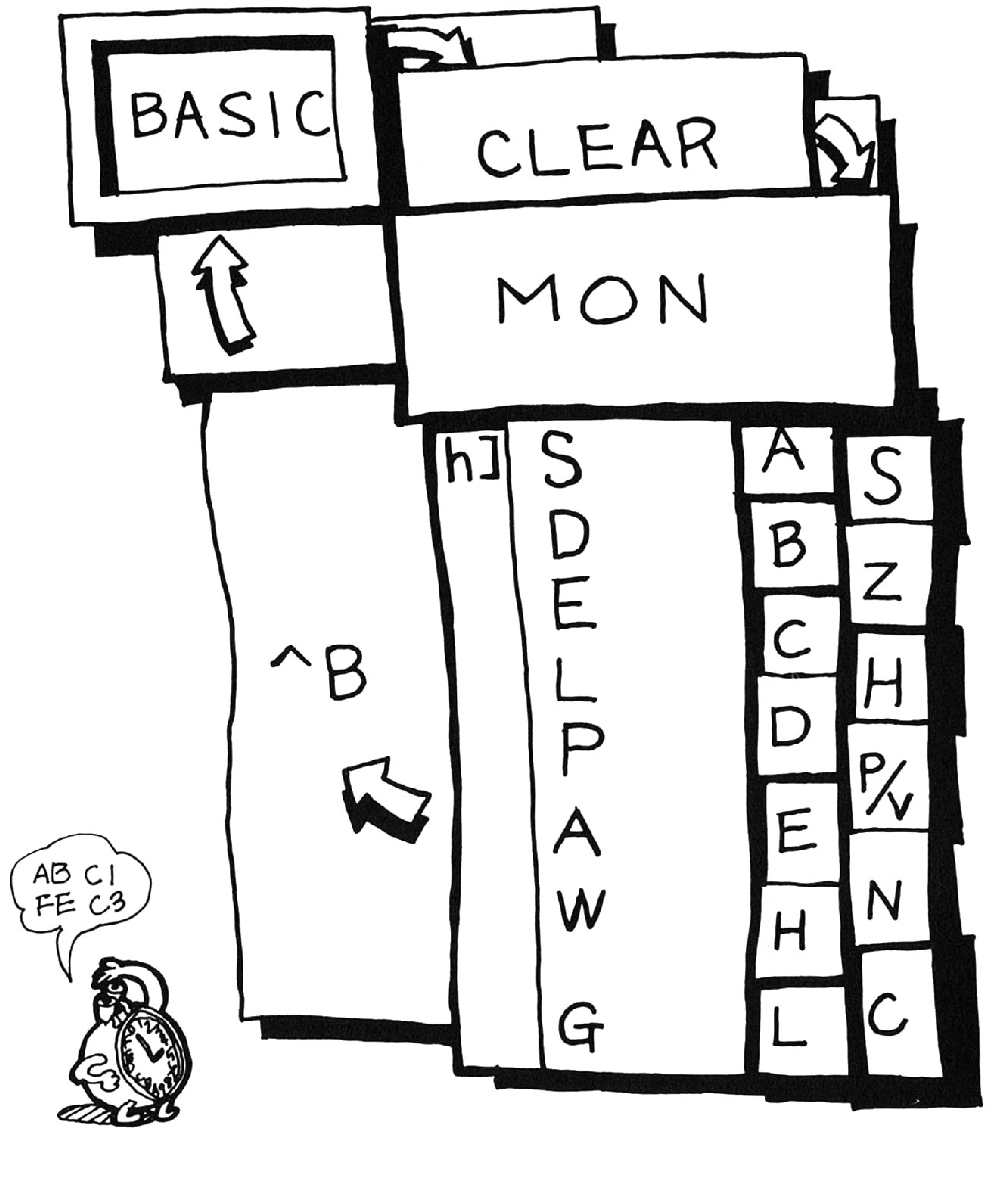
```
CLEAR ,&HB5FF⏎
```

&Hは、コンピュータに「これから入力するデータは十六進数である」と伝えています。こうして指定した$B5FF_H$のつぎの番地、すなわち$B600_H$番地からメインRAMの最後の番地である$E5FF_H$番地までが完全にユーザーエリアとなりました。これで、BASICにより内容を破壊されることはなくなります。

ただし、このことは、それだけBASICプログラムで使用できる変数エリアが減るということですから、BASICとマシン語をミックスして使う場合は気をつけなければなりません。

BASICのフリーエリアがどのくらい残っているかは、PRINT FRE(0)とすることにより確認できますが、PC-8801のバージョンやDISK-BASICとROM-BASICでも違いますし、またHow many files (0－15)?の値によっても違ってきますので、いろいろと実験してみてください。

第2章
マシン語操作の基礎



パーソナルコンピュータも商品ですから、メーカー側はつねにユーザーに注目されるよう手を変え品を変えて、話題づくりに力を注いでいます。これは企業として当然のことなのですが、車などとちがって、パソコンの場合、新製品と旧製品との差は著しいものがあります。

その是非はともかく、プログラマーにとってもすべてに対応しなければならないため、バージョンが増えれば苦労も増えるというのが実情です。この傾向は今後も続くと思われますが、これは新たにマシン語をおぼえようとしている人にとっても大変なことです。

事実、生まれてはじめて購入した機種がPC-8800シリーズのSR以降のマシンである人には、SPEEDやMODEの切り換えがなぜあんなにいくつも必要なのかわからないと思います。

そのうえ、PC-8801のモデルチェンジや性能アップに歴史があるように、マシン語プログラムにも進歩という歴史があるのです。あとから始める人ほど、おぼえなければならないことも多くなっていきます。だから、マシン語学習は早く始めたほうが得、「今が最後のチャンスです!!」なんていうコピーは大げさでしょうか……。

2-1 マシン語の入力方法

第1章を読み終えたことで、マシン語に関する基礎的な知識はついたと思いますので、これからは実践編というかたちで、マシン語を具体的に操作していくことにしましょう。

実践編ですから、必ず同じことをパソコンに向かって実行して、その結果をあなた自身の手で確認するようにしてください。

最初ですから、時には間違ったことをやってしまうかもしれません。しかし、そのすべてがマシン語に対する新たな知識となるわけですから、失敗大歓迎のつもりでやればいいのです。そのような経験こそが、マシン語修得への最大の近道といえるでしょう。

■マシン語バイエル第1番

まずは、マシン語プログラムをメモリへ入力する方法です。

といっても、実際にはまだプログラムをつくるレベルではありませんから、単に2桁の十六進数をメモリへ入力する方法といったほうが適切かもしれません。

ここでは練習として、プログラムとしてはデタラメですが、$D000_H$番地から$D004_H$番地まで、つぎのように5つの数字を入力していくことにします。

$D000_H$番地 ←----- 10_Hを入力
$D001_H$番地 ←----- 11_Hを入力
$D002_H$番地 ←----- 12_Hを入力
$D003_H$番地 ←----- 13_Hを入力
$D004_H$番地 ←----- 14_Hを入力

練習ですが、今後のために、するべき作業はキチンとしなければなりません。マシン語に入る前に、まずBASICのCLEAR文によるマシン語エリアの確保が必要でしたね。$D000_H$番地の1つ前の番地、$CFFF_H$番地を指定します。

```
CLEAR ,&HCFFF⏎
```

これで、$D000_H$番地からメインRAMの最終番地である$E5FF_H$番地までがマシン語を自由に置けるエリアとなりました。

■Sコマンド

つづいて、モニタ・モードに入ります。MONでしたね。モニタ・モードのプロンプト（h]）が表示されたら、数字を入力します。それにはS（セット）というコマンドを用います。

```
MON⏎
h]SD000        ←····SD000と入力したらつぎにスペースバーを押す
```

スペースバーを押すと、現在、そのメモリに入っている数値が表示され、ハイフンのあとにカーソルが点滅しています。そこで、入力したい数値を入れ、ふたたびスペースバーを押します。Sコマンドは、スペースバーを押していくことにより、つぎのアドレスへ連続して入力していけます。この例の場合は、スペースバーを押すたびに、D000→D001→D002→D003→D004と番地が進みます。その都度、現在の数値が表示されカーソルが点滅するので、入力したい数値を入れるのです。

■入力完了はリターンキー

入力を完了したいときは、リターンキーを押します。すると、h]の状態にもどります。

以上のプロセスを経ると、つぎの図のようになります。

```
h]SD000_00-10_FF-11_00-12_FF-13_00-14⏎   ※_はスペースを示す
h]▮
```

■入力確認はDコマンド

入力はこれで完了したはずですが、本当にメモリに書きこまれたかどうか一応確認をしてみましょう。

メモリ内容を見るには、D(ダンプ)というコマンドを用います。Dコマンドは、見たいメモリの先頭アドレスと終了アドレスを入力します。

```
h]DD000, D004⏎
D000 10 11 12 13 14
h]▮
```

間違いなく、メモリに入っていますね。

なおDコマンドは、先頭アドレスを指定するだけでもメモリの中身を表示します。ただし、この場合には、自動的に16バイトの表示をします。確かめてみましょう。

■Eコマンド

ところで、Sコマンドによる入力は、セットする数が少ない時はいいのですが、多い場合はスペースバーをいちいち押すのが面倒になってきます。そういう時は、E(エディット)というコマンドを使うと、スクリーン上でエディットしながら直接メモリに入力することができます。

```
h]ED000⏎
```

この命令を実行すると画面がクリアされ、$D000_H$番地から$D0FF_H$番地までのメモリ内容が一度に表示され、カーソルが$D000_H$番地の内容の所で点滅しているはずです。カーソル移動キーで、カーソルを希望のアドレス位置まで移動するだけで数値を入力できるので、だいぶ楽だと思います。

また、この状態だと、[ROLL UP]キーや[ROLL DOWN]キーによりアドレスを前後にスクロールさせていくことも可能になっています。

エディットモードからぬけて、h]の状態にもどるには、[STOP]キーか[ESC]キーを押してください。

```
    ↙カーソルがブリンクしている
D000 10 11 12 13 14 00 00 00 00 00 00 00 00 00 00 00
D010 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D030 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D040 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D060 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D090 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D0A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0B0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D0C0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D0D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
D0E0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
D0F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

ここまで練習してみると、第１章の最初でやったh］D0 ↵ の意味、もちろんプログラムの意味ということではなく、なにをしたのかの内容が、わかったのではありませんか。

あれは、結局、0000_H番地からのBASIC ROMの内容を見ていたのですね。0ということは、４桁で表すと0000というわけです。

それでは、どうせですから、ところかまわずいろいろなアドレスの内容も見てください。

とはいうものの、そんなことをすると、たぶん、数字アレルギーになる可能性が大かもしれません……。

2-2 もう１つのマシン語

いくらマシン語としての意味は無視するといっても、こうも数字ばかり出てくるのではおもしろくありませんね。まだ挫折した方はいないと思いますが、「自分にはマシン語は向いていない」と感じている方も少なくないかもしれません。このへんで、そろそろプログラムのための本当のマシン語を紹介しておかないと、マシン語への失望のほうが期待感よりも大きくなってしまいそうです。

■パーフェクト翻訳装置

以前、数字だらけの世界には「マシン語パーフェクト翻訳装置」で対抗できると書いてあったのをおぼえているでしょうか。

じつは、これはBASICのように、ある決まった内容に対する「お仕着せの通訳」ではなく、マシン語の数値に１対１にダイレクトに対応する翻訳装置という意味なのです。

つまり、この翻訳装置を使用すれば、マシン語を数字として扱うのではなく、意味を持った言語として扱えるということです。マシン語がこれまで無機質な数字の集まりであったことにくらべると、これはエライ違いではありませんか。

そして、このような言語のことを**アセンブリ言語**といい、これをマシン語の数字に変換してくれる翻訳装置のことを**アセンブラ**というのです。

アセンブリ言語は、実際には例の２桁の十六進数であるマシン語と１対１に対応しているわけですから、人間が扱いやすいというだけで、内容的には完全にマシン語そのものです。そのため、これもマシン語と呼ばれています。

これから、あなたがおぼえていくマシン語とは、本当はこの第三のマシン語のことだったのです（第一のマシン語＝８桁の二進数、第二のマシン語＝

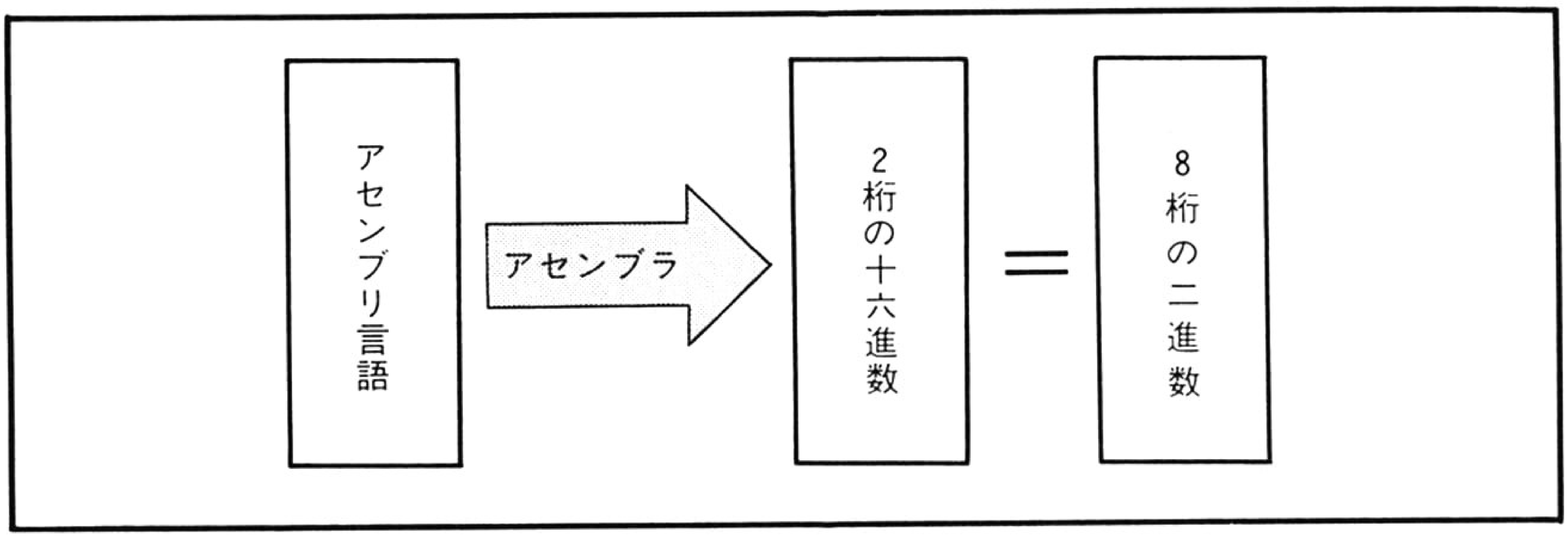


図2.1　マシン語プログラミングの手順

2桁の十六進数)。

そして、PC-8801のスバラシサも、ここで発揮されているのです。

それは、このアセンブラが、なんと本体に標準装備されているという点です。8ビットパソコンで、これは他に例を見ない機能です。

ただ、この標準装備アセンブラが、アセンブラとして100パーセント満足できるだけの能力があるかというと、正直なところ力不足といわざるをえません。しかし、マシン語入門用として、またPC-8801をうまく活用する意味においても、利用しない手はありません。

では、本格的なアセンブラにくらべて、どのような違いがあり、欠点があるのでしょうか。気がかりかもしれませんが、それについては、ここでは触れないことにします。

いまは、それを知ることよりも、このアセンブラを使いこなすことが先決ですし、その結果、不満や不便さを感じて、はじめて、つぎのステップへ進んでも決して遅くはないからです。

そこまで言っても、どうしても気になる方は、本書の最後の章に書いてありますからそこを先に読んでみてください。

とにかく、ここでは、アセンブリ言語とはどんな形をしているのか、のぞいてみることにしましょう。

■Lコマンド

すでに第1章でもやったことですが、これにはL(ディスアセンブル)とい

うコマンドを用います。ディスアセンブルとは、アセンブルされたものを分離してもとにもどすわけですから、2桁の十六進数の形にさせられた命令がそうなる前のアセンブリ言語の形にもどるわけですね。

```
h]L0⏎
0000    F3          DI
0001    31 E1A0     LXI   SP, E1A0
0004    C3 3BE5     JMP   3BE5
0007    00          MOP
0008    7E          MOV   A, M
000A    BE          XTHL
000B    23          CMP   M
000C    E3          XTHL
000D    C2 0393     JNZ   0393
h]■
```

この点線枠で囲まれた部分が、アセンブリ言語で書かれたマシン語プログラムです。

左はしの十六進の数字は番地で、見てのとおり、これは0000_H番地からのマシン語プログラムです。ところが、よく見るとそのアドレスが飛んでいます。このことは、命令により、使用するバイト数が違うことを示しているのです。つまり、命令には、1バイトで表現できるものがあれば、3バイト必要とするものもあるというわけです。

■ニモニック

このアセンブリ言語による1つひとつの命令のことを、**ニモニック**といいます。

ニモニック(mnemonic)とは、記憶を助ける工夫というのがもとの意味です。たとえば「ウグイス鳴くよ平安京」といって、京都に都がおかれた年が794年であったことをおぼえた経験があるでしょう。このような記憶術（法）をニモニックといいます。

転じて、アセンブリ言語の記憶用コードということになったわけですが……しかし、このニモニックを見ても、いったいどこが人間にわかりやすいのでしょうか。数字のときと大差ないように感じるかもしれません。

でも、これはプロレスに興味のない人には、IWGP とか NWA などといってもわからないのと同じことで、ある英単語を省略して作成した記号ですから、もともとの意味がわからなければ解読は困難というだけのことです。

それよりも、L コマンドにより表示された結果は、左からアドレス、その内容である 2 桁の十六進数、および、それに相当するニモニックである、ということが理解できただけで、われわれのマシン語に対する理解は、第 1 章の時点より 1 ランク上がったと確認できるではありませんか……。

なお、L コマンドによるディスアセンブルは、D コマンドと同じように、アドレスの最初と最後を指定することにより、連続して見ることができます。

途中停止は CTRL ＋ S です（コントロールキーと S のキーを同時に押すことを示します）。これは BASIC と同じですね。

■Pコマンド

また、マシン語では、実行結果をプリンターに出力するには P（プリント）というコマンドを用います。h] P ⏎ とすると、プロントの表示が h) に変わり、これ以降は L コマンド、D コマンドによる指示が、プリンターに対しても有効となります。

h] の状態にもどるには、再度 P コマンドを入力します。

L コマンドで注意すべき点は、開始アドレスを命令の始まりのアドレスに合わせないと、結果が狂うということです。これは、2 バイト以上使用する命令をディスアセンブルするときに起こります。

ただし、ここでもプログラムの内容を気にせず、いろいろと実験することのほうが大切です。だから、あまり細かいことは考えずに、このコマンドの使い方の練習くらいのつもりで、アチコチのアドレスの中身をのぞいてみてください。

2-3 ニモニックによるマシン語入力

マシン語の命令とは、どんな内容だと思いますか。
「リアルタイムゲームがマシン語でつくられているということは、たぶんゲームに関する便利な命令がたくさんあるはずだ」なんて、ウキウキしている人もいるかもしれませんね。

これが、じつは、どうにもガッカリするようなことしかできないのです。

■マシン語命令を分類すると

簡単に分類すると、つぎのようなつまらない内容になってしまいます。

(1) 数字をもてあそぶ――つまり足したり、引いたり、論理演算などをする命令

(2) BASICふうにいうならば、GOSUB文やGOTO文に当たる命令

(3) PC-8801本体、および周辺装置を制御したり、調べる命令

あまりにつまらなくて、マシン語への意欲も半減といったところでしょう。

しかしBASICも、しょせんはこれだけの命令からできている、という事実を思いだしてください。ここから、どのようなルーチンに仕上げるかは、まさにプログラマーの腕とアイデアしだいといえるのです。

これが、マシン語がパズルたるゆえんです。1行1行の命令がわかったところで、マシン語が理解できたとはいえないのです。またその逆に、全体の意味がわかっても、1行1行の意味がわからなくては、これまたマシン語を理解できたとはいえません。

前者は、城を見ないで石垣ばかり見ているのと同じことであり、後者は城ばかりを見て石垣を忘れているようなものです。これでは、どちらも城を造ることはできません。

といっても、その前に石垣の積み方がわからなければ、石の前でウロウロ

するばかりになってしまいます。大きな城にするのか、小さな城にするのか、あるいは形をどうするのか、これらはすべて石垣が積めるようになってからの話ですね。

ここでやることは、早い話が石積みの練習、つまりニモニックの入力方法のマスターです。その石、いや命令がどんな意味を持つかについては、この際、無視することにしましょう。どうせ、前述したような内容でしかないのですから、それほど気にすることはありません。

■マシン語バイエル第2番「ニモニック」

石積み練習は、Lコマンドで見たばかりの0000_H番地から$000D_H$番地までのプログラムを、$D000_H$番地から順に入力してみるという簡単なものです。

これはマシン語の入力ですから、まずBASICのCLEAR文（CLEAR, &HCFFF ⏎ ）によりマシン語エリアの確保を行い、続いてA(アセンブル)というコマンドにより、ニモニックでの入力をします。

ところでアセンブルという言葉は、すでにアセンブラという名詞形でたびたび使ってきていますが、本来、機械などの部品を集めて組み立てるという意味で、コンピュータ用語では、ニモニック記号言語（アセンブリ言語）を機械語（マシン語）に変換することをいいます。これで、この章の第2節でアセンブラのことを「翻訳装置」といったわけがわかってもらえるでしょう。

では、入力してみましょう。

```
h]AD000⏎
D000          ▌ ←‥‥‥カーソルがブリンクしている
```

このカーソルがブリンクしている地点から、前節でLコマンドで見た通りのニモニックを打ち込んでいけばいいのです。すなわち、

```
h]AD000⏎
D000          D1⏎
```

とするわけです。するとこのニモニックはすぐにアセンブルされて画面に

表示され、さらにつぎの番地が表示されます。ここでまたカーソルがブリンクし、つぎのニモニックの入力を促します。

この作業でアセンブルされたニモニックは、メモリにその場で格納されていきます。

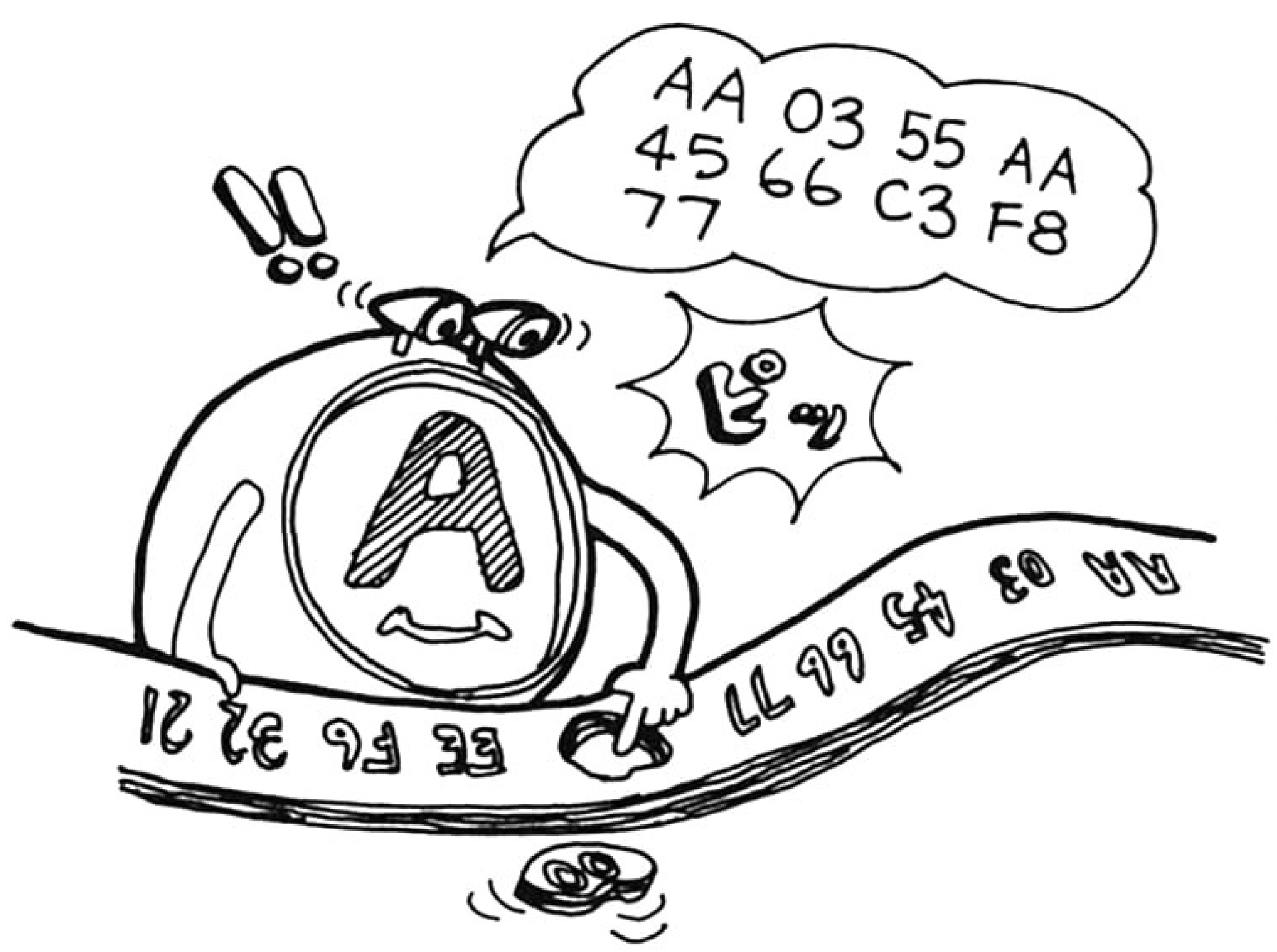
```
h]AD000⏎
D000    F3      DI
D001            ▌ ←……カーソルがブリンクしている
```

これを繰り返し、予定したすべての入力が完了したならば、リターンキーか STOP キー、または CTRL ＋ C を押すことにより、h] の状態にもどります。

■エラー訂正はAコマンド

もしもこの入力の途中で間違いがあると、ピーッというビープ音とともにクエスチョンマーク（？）が表示され、画面は h] の状態にもどってしまいます。そういう場合には、アドレスなしで A コマンドを入力すれば(h]A ⏎)、間違いのあったアドレスが表示され、その番地から入力を再開することができます。

本書で作成するマシン語プログラムは、すべてこの A コマンドによる入力となりますので、ここで失敗をしたり、いろいろと経験をしてみてください。



2-4 マシン語と変数

これまでに、モニタの使い方についてかなり練習をしてきましたから、もうマシン語操作に対する下準備は整ったといってもよさそうです。やっとマシン語本体にたどり着いたという感じですが、実際にプログラミングをする前に、ニモニックについても予備知識があるとあとでグッと楽になります。

すでに、ニモニックに対する漠然とした知識はつかめているかもしれませんが、具体的なことについてはまだゼロの状態です。そこで、プログラムには必要不可欠の変数について、ここで基礎的な知識を得ることにしましょう。

■レジスタ

BASICでプログラムを組む場合、だれでも変数を自由勝手に作ることができましたが、マシン語には変数と呼ばれるものはありません。これではプログラムの組みようがなさそうですが、その代わり、マシン語には、**レジスタ**という、いわば「標準仕様の変数」が用意されています。

A　　B C　　D E　　H L

この7つが、変数として使える基本的なレジスタです。

スーパーマーケットで支払いをする場所をレジといいますが、これは、キャッシュレジスターを省略した呼び名ですね。日本語でいえば金銭登録器ですが、ここでいうレジスタも語源は同じです。BASICの変数とは、記憶する場所のことでしたが、レジスタもまた一時的に記憶しておく場所という意味で用いています。

このほかにもレジスタと名のつくものはあるのですが、それらはマシン語に慣れてからの話で、まずはこの7つのレジスタをフルに活用できるようにならなければなりません。

各レジスタが持てる値は、メモリと同じく1バイトの数値、すなわち00_H～FF_Hのあいだの2桁の十六進数で、この範囲を越える数値の計算（加減算）をさせた場合は、時計の針と同じようにループしてしまいます。

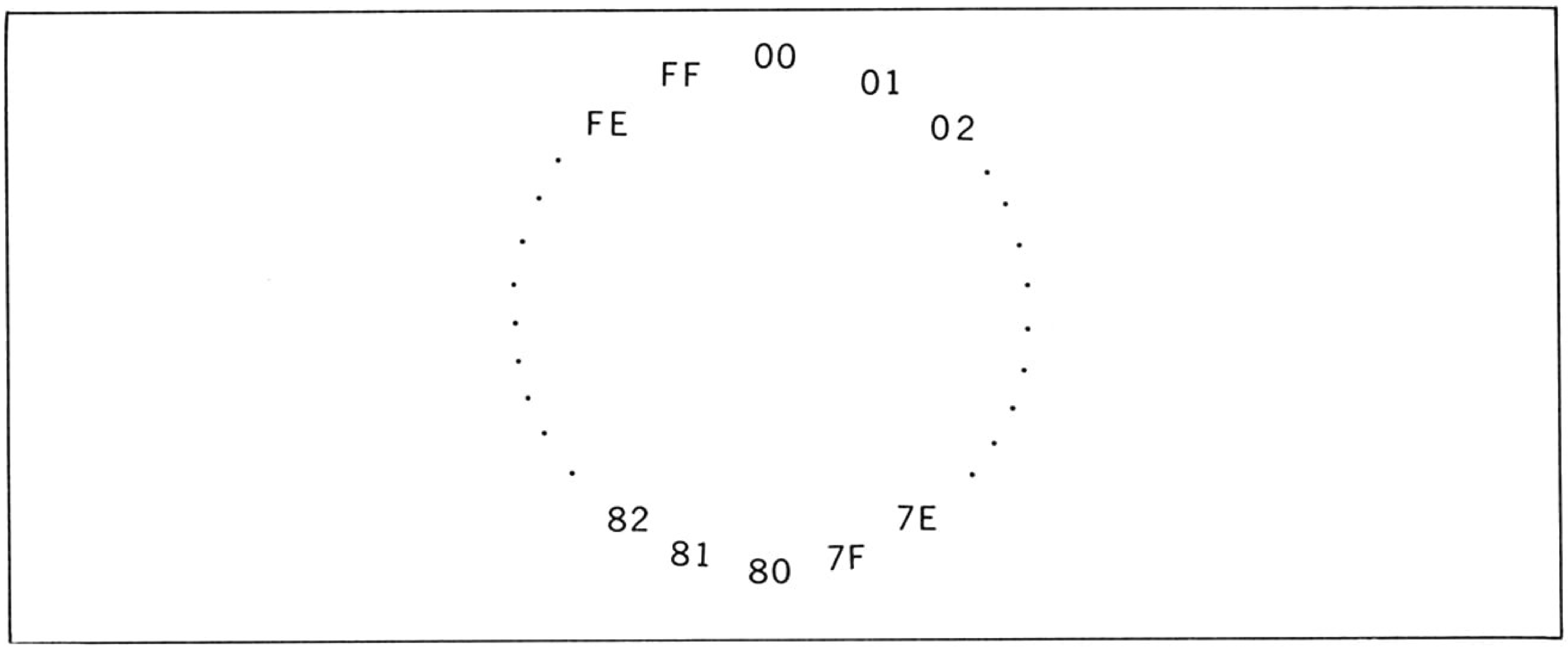


図2.2　レジスタが持つことのできる数値(00_H～FF_H)

これらのレジスタは一見すると同格のように見えますが、じつはそれぞれに特徴を持った個性派集団なのです。

■アキュムレータ

たとえば、Aレジスタは計算に関する命令が多いので**アキュムレータ**といわれ、レジスタ同士の計算や各種演算に対する命令は、ほとんどがこのAレジスタを介して行われることになっています。

また、残りの6つのレジスタは、単独で用いるばかりでなく、BC、DE、HLのようにそれぞれペアを組んで2バイトの数値（0000_H～$FFFF_H$）を扱うこともでき、その際には**ペアレジスタ**と呼ぶことになっています。

これ以上の細かな特徴については具体的な例が出てきたときにおぼえるとして、いったいレジスタが特徴を持っているとはどういう意味なのか、もう少しハッキリさせないことには消化不良になってしまうかもしれませんね。

例をあげて説明をすることにしましょう。

B＝2、C＝3とします。この場合に、BASICふうにいうならばB＝B＋Cという代入文を実現したいとします。

BASICであれば、このままB＝B＋CとするとB＝5と簡単になってしまいますが、あいにくマシン語にはB＋Cという命令がありません。あるのはAレジスタとなにかを加算する命令、つまりA＋xだけなのです。

このため、B＝B＋Cを実現するには、つぎのような手順を踏むことになります。カッコ内はBASICふうの表記です。

(1) AレジスタにBレジスタの値を代入（A＝B）

(2) AレジスタとCレジスタの値を加算（A＝A＋C）

(3) BレジスタにAレジスタの値を代入（B＝A）

つまり、レジスタの特徴というのは、それ自身の特徴ではなく、マシン語の命令によって強制された特徴といえるものなのです。

といっても、今ここですべてのレジスタの特徴をおぼえようというのではないので、なにもあわてることはありません。ただ、レジスタはどれも同じレベルではないということさえ頭に入っていれば、あとは自然にそのレジスタのカラーがわかるようになってきます。要は、習うより慣れろの精神でマシン語をいじっていればいいのです。

■レジスタが7つでは不足しないか

さて、レジスタに関する理解が深まるにつれ、数が7つしかないのは変数として足りないのではないかという心配が生じてくるかもしれません。これは、確かに多いほうが便利ですし、実際、複雑なプログラムや速度を極端に追求する場合には、本書には出てこない特殊なレジスタも使用しなければならなくなってきます。

しかし、普通はこの7つをヤリクリすれば、なんとか間に合うものなのです。

というのは、マシン語プログラムでは、変数を保存するためにレジスタを使用するのではないからです。レジスタの本来の目的は、数値の計算をするのに一時的に数値を記憶するためで、これを保存に使ったのではアッという間にレジスタはなくなってしまいます。

では、変数はどうやって保存するかというと、メモリにしまっておけばい

いのです。計算の終わった変数をメモリにしまえば、そのレジスタは別の用途に使用可能になるし、ふたたびその変数が必要ならば、メモリの値をレジスタに読み出せばいいというわけです。

こういう感覚は BASIC ではなじみが薄いと思いますが、マシン語においてはいちばん基本的なテクニックになっています。

■ワークエリア

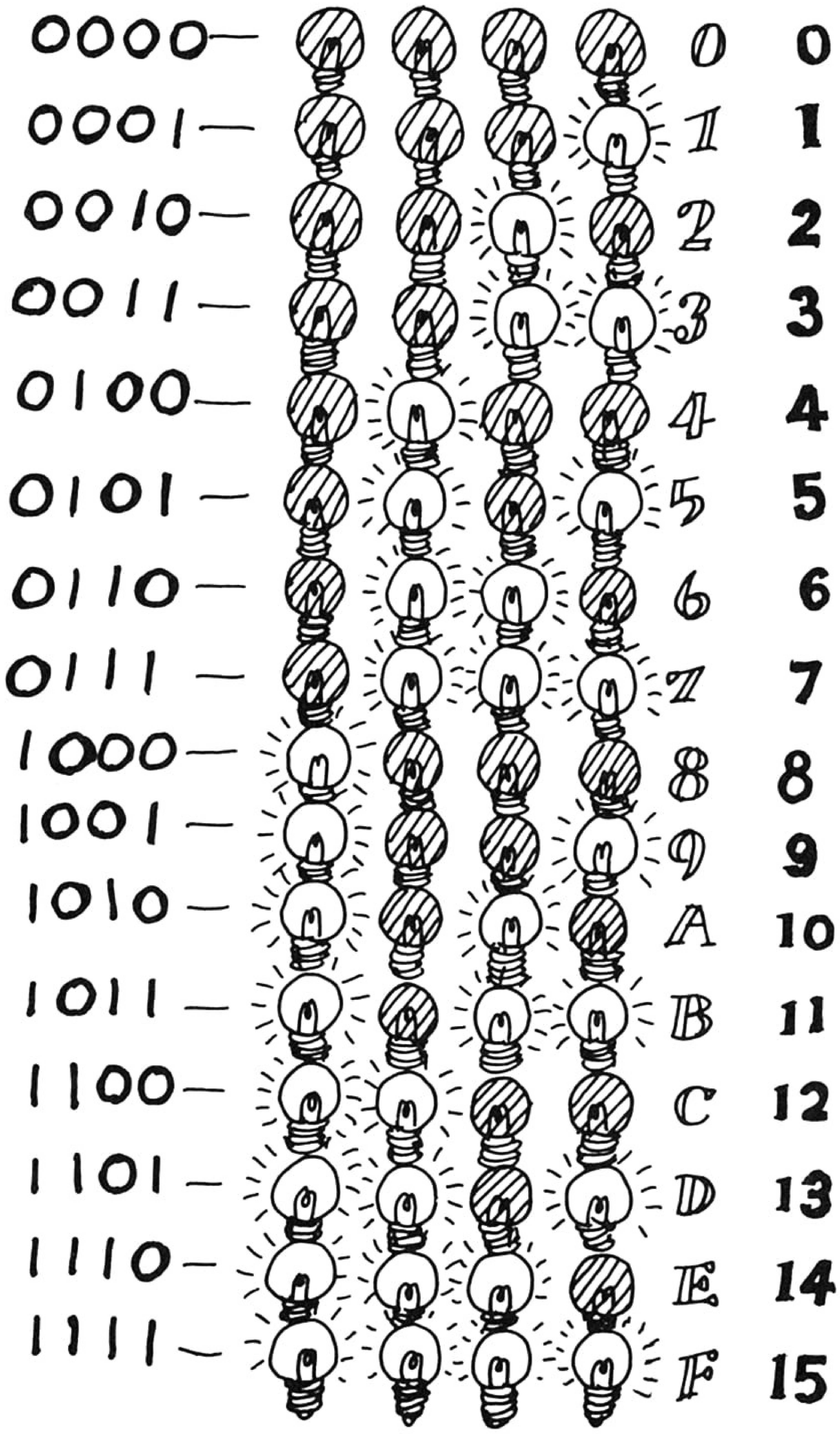
このように、メモリがプログラム以外の目的に使用された場合、そのメモリエリアのことをワークエリアといいます。

図1.4において $E600_H$ ～ $FFFF_H$ が N_{88}-BASIC ワークエリアとなっていますが、これは BASIC ROM が使っているワークエリア（実際には全部がワークエリアというわけではない）という意味です。レジスタが限られている以上、マシン語プログラムとワークエリアは切っても切れない関係にあるといえます。

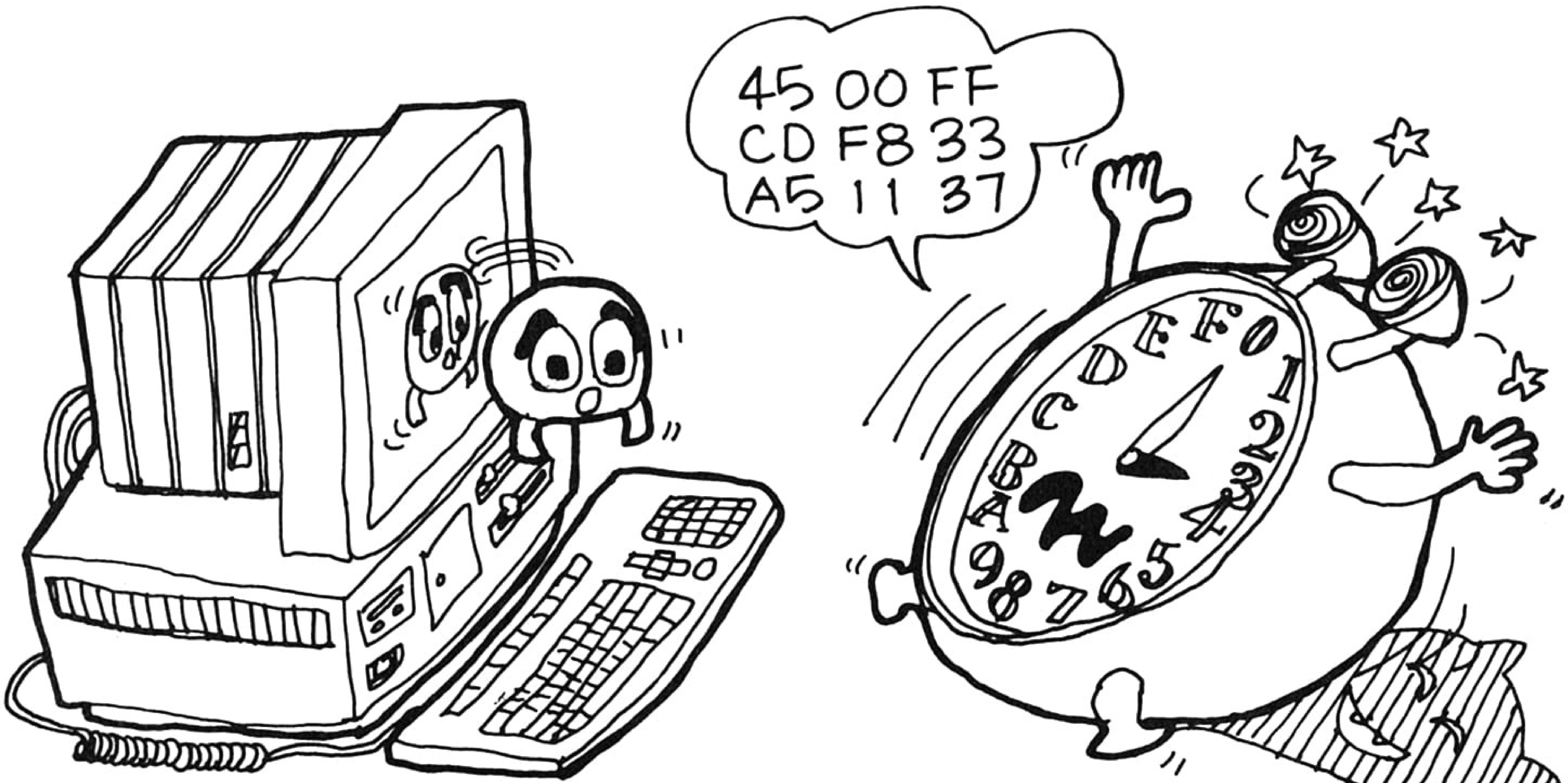
レジスタとメモリとの関係をうまく保つためには、まだまだ各レジスタについてその個性、特徴を知る必要があります。これはニモニック命令を使いこなすということですから、１つひとつ経験を積んでいくしかありません。

つぎの第３章からは、いよいよ本格的（？）にニモニックによるプログラミングが始まります。どうか、あせることなく、確実にステップを踏んでいってください。

《豆知識》　4ビットの二進数・十六進数・十進数



第3章
実践マシン語プログラミング



ここまで本書を読まれてきた方にも、はじめてパソコンに触れた時があったはずです。BASICを勉強したてのころを思いだしてみてください。恐らく、1つひとつの命令をダイレクトに実行しては、その結果を目で確認するという作業をしていたのではないでしょうか。とくにグラフィックス関係の命令を実行させてうまくいった時などは、ある種の感動すらおぼえたことでしょう。

マシン語でもこれと全く同じことがいえます。自分の作成したキャラクターパターンが画面を動いた時などは、マシン語ができたということ以上にうれしくなってしまうものです。

さて今から、その感動の日を目指してプログラムに励むわけですが、BASICをおぼえた時と同じように、必ずプログラムを実行し、その結果を自分の目で確認することを忘れないでください。感動と同時に、それがマシン語マスターのベストウェイだからです……。

3-1 はじめてのマシン語プログラム

どんなに大きな川でも、もとをたどれば一滴の湧き水から始まるように、ここで作成するマシン語プログラムは、すべてのグラフィックスのまさに原点となるものです。640×200ドットの画面いっぱいに広がるグラフィックスも、かわいいキャラクターパターンも、これらのプログラムをもとに発展していったものといえるのです。

そんな素晴らしい可能性を秘めた貴重なプログラムが、あなたがはじめて作成するマシン語プログラムとは、なんという幸運に恵まれたことでしょうか。「ラッキー！」なんて、Vサインを出してしまいそうです。なにはともあれ、早速、プログラムの入力、そして実行による結果確認をしてみましょう。

■さあ入力

このプログラムのスタートアドレスは$BD00_H$番地になっていますが、CLEAR文は、$B7FF_H$番地（CLEAR，&HB7FF ⏎）で宣言してください。これは、おわかりかと思いますが、今後、プログラムのたびにいちいち変更しなくてもいいようにするためです。この実行により、マシン語用エリアとして$B800_H$～$E5FF_H$番地が確保できたことになります。

では、Aコマンドを使い、つぎのプログラム（**LIST3-1**）を入力してください。Aコマンドによる入力のしかたを忘れてしまった人は、32ページにもどってください。

■Gコマンド

プログラムの実行にはG（ゴー）というコマンドを用いますが、これは、BASICでいうならば、RUNよりも、GOTO文でプログラムをスタートさせるほうに近いものです。その理由は、マシン語における変数であるレジスタの値が、スタートによってクリアされずに保存されたまま実行開始されるか

LIST 3-1

```
BDØØ    F3           DI
BDØ1    3E FF        MVI   A,FF
BDØ3    D3 5C        OUT   5C
BDØ5    32 CØØØ      STA   CØØØ
BDØ8    D3 5F        OUT   5F
BDØA    FB           EI
BDØB    FF           RST   7
```

らです。コマンドのあとにスタートアドレスを指定します。

■マシン語プログラムのセーブ法

実行に移る前に、プログラムを再確認するのは当然のことですが、念のために必ずセーブしておく習慣をつけましょう。マシン語のセーブ方法はテープとディスクで違いますから、間違わないようにしてください。

(1) テープにセーブする場合（ファイル名はなくてもかまわない）

```
MON⏎
h]Wファイル名，セーブ開始アドレス，セーブ終了アドレス⏎

(例) LIST 3-1の場合
     h]WLIST1, BD00, BD0B⏎
```

(2) ディスクにセーブする場合

```
BSAVE "ファイル名", セーブ開始アドレス，セーブするバイト数⏎

(例) LIST 3-1の場合
     BSAVE "LIST1", &HBD00, &HBD0B − &HBD00 + 1⏎
```

セーブが終われば、実行です。

```
h]GBD00
h]▮
```

アッという間に実行完了ですが、画面のどこかに変化が起きているはずです。

よくわからない時はBASICにもどり、テキスト画面をクリアしてカーソルを下のほうへ移動してみてください。画面の左上に、ほんの数ミリの青い線がありませんか。これこそが、すべてのグラフィックスの基礎となる記念すべき線なのです。決してゴミなんかではありませんから、もっとオーバーに感動してもいいくらいです。

■このプログラムで実行したこと

1行1行の命令については、つぎの節でジックリ理解してもらうことにして、ここでは全体としてどのようなことを実行させたのかを認識してもらうことにします。

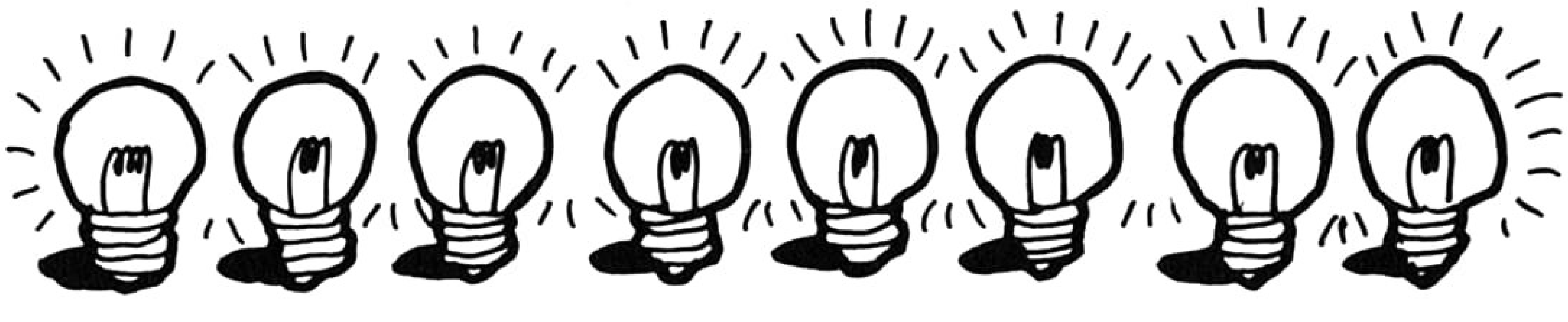
もう一度、**図1.5**を見てください。

このプログラムでやったことは、ここにあるグラフィックV.RAMのブルー面にバンク切り換えをして、C000$_H$番地にFF$_H$を入れるという作業なのです。そして、その結果が画面左上の青い線というわけです。さらにつけ加えるなら、この青い線の長さは、グラフィック画面の横640ドットのうち、8ドットに相当しているのです。

この関係は、FF$_H$を二進数に直すとより明確になります。

$$FF_H = 1111\ 1111$$

すでに理解された方もいると思いますが、グラフィックV.RAMとはグラフィック画面を表示するための特殊なメモリのことで、1メモリで8ドットのON/OFFが管理できるようになっています。つまり、各ドットをONにした状態が1、OFFのままなら0というわけで、ONならば明かりがつく、OFF



ならばつかないということになります。

このように、グラフィックV.RAMに入れた数値のビットイメージが、そのままグラフィック画面のドットでカラー表示されるということです。

グラフィックV.RAMには、ブルー面用、レッド面用、グリーン面用と3面あり、それぞれ同じアドレスでそろっています。ですから、その組み合わせにより、カラーコード0～7のドットを容易に作ることができることになります。

この面のことをプレーンといいますが、それぞれのプレーンとカラーコード（番号）の関係は右の表のようになっています。

グリーン面	レッド面	ブルー面	カラー番号
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

（注）各面の0/1はビットイメージの0/1を表す．

図3.1　カラー番号とビットイメージの関係

■グラフィック座標とG.V.RAM

さて、グラフィック画面は640×200ドットですから、グラフィック座標とグラフィックV.RAMにおけるアドレスとの関係も知っておく必要があります。C000$_H$番地が画面左上であることはわかっているわけですが、残りも図3.2のように素直にならんでいます。FE80$_H$～FFFF$_H$番地は、なにも使用されていない空エリアということになります。

座標	0	128	256	384	512	639
0	C000	C010	C020	C030	C040	C04F
1	C050	C060	C070	C080	C090	C09F
2	C0A0	C0B0	C0C0	C0D0	C0E0	C0EF
⋮	⋮	⋮	⋮	⋮	⋮	⋮
197	FD90	FDB0	FDB0	FDC0	FDD0	FDDF
198	FDE0	FDF0	FE00	FE10	FE20	FE2F
199	FE30	FE40	FE50	FE60	FE70	FE7F

図3.2　グラフィック座標とグラフィックV.RAMの関係

■画面を拡大してみると

ところで特殊な虫めがねを使って、さきほどのプログラムを実行した結果の画面から$C000_H$番地付近を拡大してみましょう。そうすると、ブルーの線の描かれている状態がハッキリと確認できると思います。

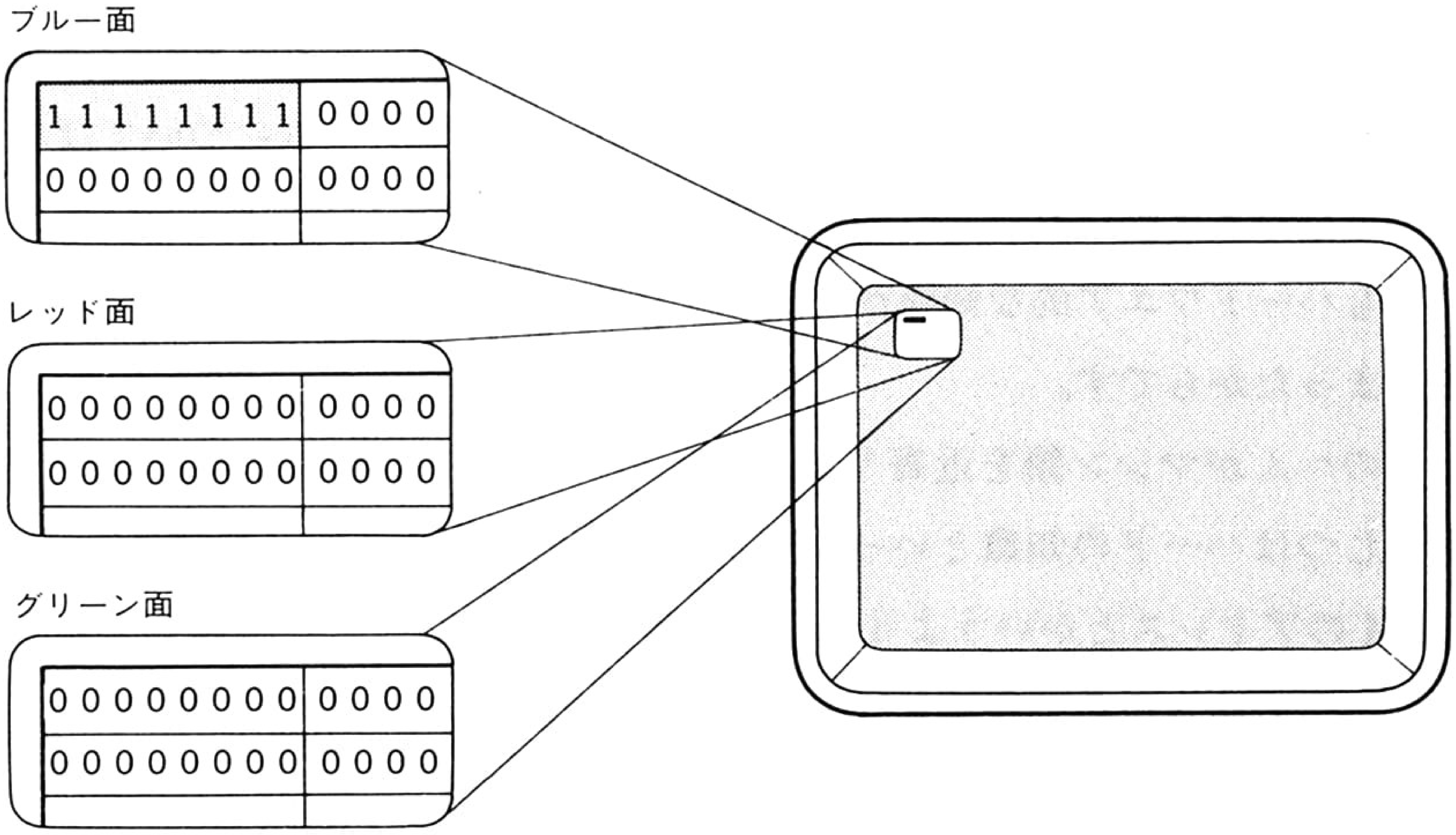
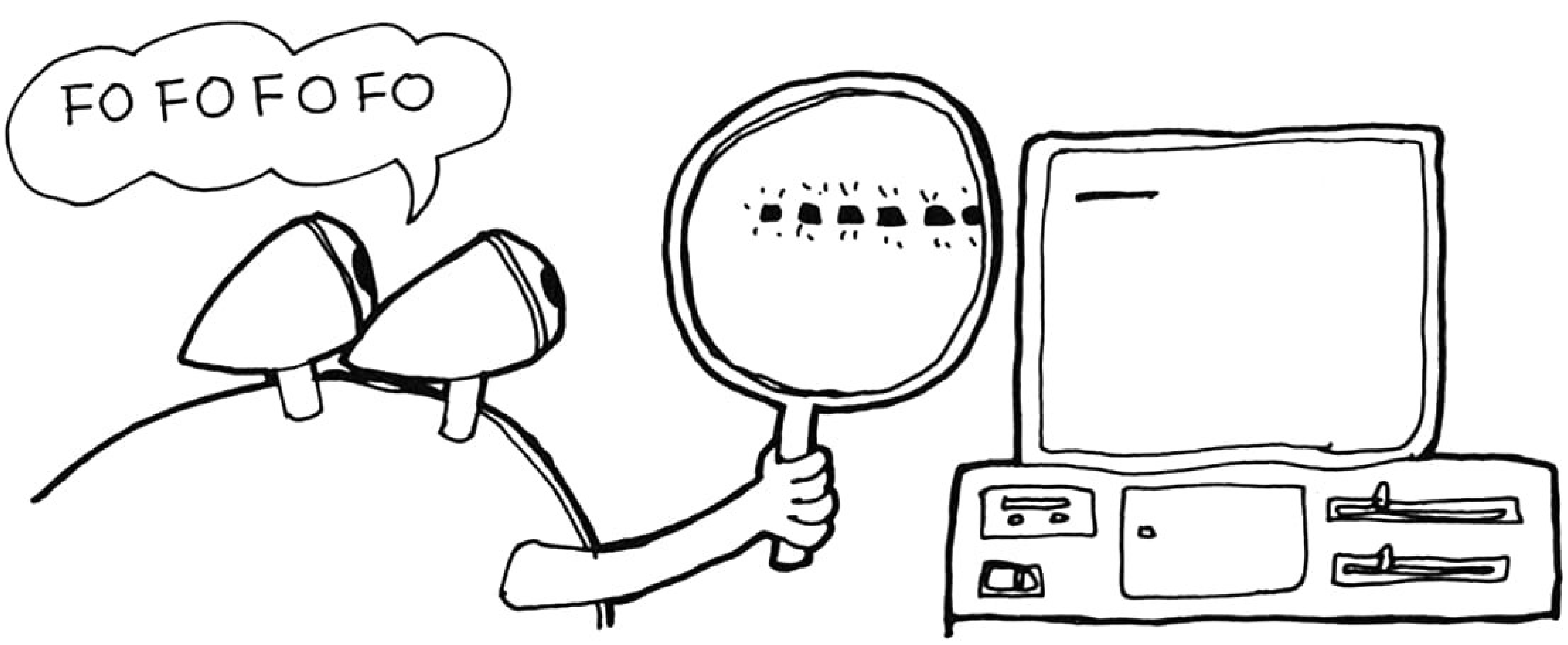


図3.3　$C000_H$番地付近の拡大図

まず3つのプレーンのうちブルー面が使用されています。$C000_H$番地にFF_Hを入れた結果、各ビットが1になり、それがドットに対応して8ビット分のドットを点灯しているのです。ドットとは点のこと、点のつながりは線ですから、ここでは8つの点が青い線として見えるわけです。



3-2 ニモニックの解析

LIST 3-1を実行すると、画面左上にブルーの線が引かれるという過程と結果は一応の納得がいったと思いますが、肝心のプログラムについてはなにも話さないままに終わってしまいました。これは、マシン語の説明にはどうしてもハードウエア的な知識が必要になってくるため、仕方なく後回しにしてしまったからです。

このへんがマシン語を近寄りにくいものにしている最大の原因なのですが、じつはハードの知識といっても、バンク切り換えとかグラフィックV.RAMのアドレスとかいうように、PC-8801の構造的なことに関してですから、実際には文科系の方でもビクビクするほどのものではないのです。

■ニモニックの意味

それでは、**LIST3-1**に出てきたニモニックについての解説をしましょう。

先にふれたように、ニモニックは基本的には英単語の省略形で、しかも使われる単語の数はわずか50前後しかありません。

これを単語別に分類したインストラクション表が巻末に付録として載っていますから、おもにどのような単語が使用されているか、あとで確認してみてください。もとが同じ単語でも、命令の中身によって若干省略形が違っていますが、本来の単語の意味とつづりが想像できるようになると、あとはイモヅル式といってもいいくらい容易にわかってくるものです。

ニモニックをおぼえるポイントは、個々の内容について理解することよりも、まずは省略前の単語とその意味を結びつけるようにするといいでしょう。

■オペランド

ニモニックによっては、たとえば「～に～を代入せよ」というように目的語を持つものがあります。「～に」「～を」にあたる部分がそうですが、これ

らの目的語のことを**オペランド**といいます。オペランドが2つある場合は、前者を第1オペランド、後者を第2オペランドといい、あいだに「,」を入れて両者を区別します。また、オペランドにレジスタがある場合は、ごく一部のニモニックを除いて、その他のレジスタに対しても同じ書式が通用するようになっていますので、頭に入れておいてください。

さて、お待たせしました。それでは1つひとつの説明に入ります。

ニモニック （読み方）	説　　明
DI （ディスエーブル インタラプツ）	Disable Interruptsの略で、割り込み禁止のこと。キー入力などは、プログラム実行中でも常に割り込みによりスキャンされているが、このためのプログラムの一部は$E600_H$番地以降のワークエリアに入っている。そのため、バンク切り換えをするとこのプログラム自体も切り離されてしまい、割り込みがかかると同時に暴走することになる。そこで、バンク切り換えをするプログラムでは、切り換える前に必ず割り込みの禁止をしておかなければならない。
MVI A, FF （ムーブ イミーディエート）	アキュムレータ（Aレジスタ）にFF_Hを代入する。MVIとはMoVe Immediateの略で、Moveは代入する、Immediateは変数の値ではなく即値（数値）という意味になる。このAとFFがオペランドである。2つあるから、カンマで区別している。
OUT 5C （アウト）	グラフィックV.RAMのブルー面にバンク切り換えをする。OUT命令とは、プリンターとかCRTなどと結ぶ各種の出力ポートにアキュムレータの値を出力する命令。$5C_H$という出力ポートはグラフィックV.RAMのブルー面にバンク切り換えをするポートで、アキュムレータの値はなんでもよい。ちなみにレッド面の出力ポートは$5D_H$、グリーン面は$5E_H$である。

STA C000 （ストア アキュムレータ）	アキュムレータの値を $C000_H$ 番地にストアする（入れる）。STA は STore Accumurator の略。Store とは保管するとかしまうという意味だが、コンピュータ用語としてはメモリにデータを入れておくという意味に用いている。
OUT 5F	バンク切り換えにより、$C000_H$ 番地以降をそれまでのグラフィック V. RAM ブルー面からメイン RAM にもどす。
EI （イネーブル インタラプツ）	Enable Interrupts の略で、割り込みの許可を意味する。この命令が必要なのは、割り込み禁止のまま BASIC にもどると、命令によっては正常に動作しなくなるからである。
RST 7 （リスタート）	プログラムの実行を終え、h]の状態にもどる命令。RST は ReSTart の略で 0 ～ 7 まで 8 種類があるが、あまり使用されることはない。この RST 7 も実際のゲーム等では h] でプログラムを止めることはないので、おもにプログラムのテストやデバッグ用として使われる。

ニモニックの説明中に、グラフィック V.RAM のレッド面とグリーン面にバンク切り換えをする方法がありましたが、これを実験により確認してみましょう。**LIST3-2**を実行してみてください。

このプログラムは前のプログラムとくらべて、バンク切り換えの回数が多くなっただけの違いなので、とくに説明はいらないと思います。実行させると、さきほどのブルーの線が変化して今度は白くなっているはずです。これも、**図3.1**にあるカラー番号とプレーン（面）の関係から、正しく変化していることがわかります。今度は、アドレスのほうをいじってみましょう。**LIST3-3**のプログラムです。

グラフィック画面を消去してから**LIST3-3**を実行させると、左上に青・赤・緑の線が描かれるはずですが、**LIST3-2**を実行したままだと白・赤・緑になってしまいます。これは、グラフィック V.RAM というのは、おのおのが独立

していることを示すもので、青にしたければレッド面・グリーン面を消去しなければならないということです。

LIST 3–2

```
BD00    F3          DI
BD01    3E FF       MVI   A,FF
BD03    D3 5C       OUT   5C
BD05    32 C000     STA   C000
BD08    D3 5D       OUT   5D
BD0A    32 C000     STA   C000
BD0D    D3 5E       OUT   5E
BD0F    32 C000     STA   C000
BD12    D3 5F       OUT   5F
BD14    FB          EI
BD15    FF          RST   7
```

LIST 3–3

```
BD00    F3          DI
BD01    3E FF       MVI   A,FF
BD03    D3 5C       OUT   5C
BD05    32 C000     STA   C000
BD08    D3 5D       OUT   5D
BD0A    32 C001     STA   C001
BD0D    D3 5E       OUT   5E
BD0F    32 C002     STA   C002
BD12    D3 5F       OUT   5F
BD14    FB          EI
BD15    FF          RST   7
```

つぎへ進む前に、アキュムレータの値やグラフィックV.RAMのアドレスをいじって、いろいろとテストしてください。何度も入力することにより、ニモニックも一緒におぼえられます。

3-3 青い正方形

グラフィック画面に対する知識はかなり深まったと思いますが、今度はプログラムのレベルをもう少し上げて、コンピュータらしく、判断能力を持った作業をさせてみることにしましょう。といっても、難しい人工知能のことではなく、BASICでいえばIF文で条件分岐をさせる程度のことです。

■条件分岐

条件分岐というのはやさしそうに思えるかもしれませんが、じつは大変奥の深いテーマで、つきつめるとプログラムのアルゴリズムにまで関連してくるほどです。実際、マシン語プログラムにはたくさんの条件分岐が出てきて、まるで「条件分岐のかたまり」といっても過言ではないくらいです。ただ、ニモニックの総数から判断してもわかるように、種類のほうはそう大した数ではありません。それだけに、条件分岐をどのように使うかは、まさにプログラマーの腕の見せどころであり、結局はアルゴリズムの善し悪しにもなってくるわけです。

■好きな位置に図形を描く

さて、これから作成するプログラムですが、前書きのわりにはいたってシンプルでやさしいものとなっています。内容は、画面の任意の位置（といってもグラフィックV.RAMのアドレス単位）に32×16ドットの青い正方形を描くというものです。

まずは、正方形の表示手順を図によって見てみましょう。番号の書いてあるマスの1つひとつが、グラフィックV.RAMのアドレスに相当しています。

表示の手順は、非常に素直に正方形の左上から1ラインずつ下へ線を引いていくようになっていますが、これを実際にプログラム化していくにはどう

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16
17	18	19	20
21	22	23	24
25	26	27	28
29	30	31	32
33	34	35	36
37	38	39	40
41	42	43	44
45	46	47	48
49	50	51	52
53	54	55	56
57	58	59	60
61	62	63	64

＊グラフィックV.RAMに対し番号順にFF_Hを入れていく

図3.4　32×16ドットの正方形の描き方

したらいいか、わかりやすいようにフローチャートにしてみましょう。

■処理の流れ

このフローチャートの中で使用されているB、Cという変数は、プログラムでもそのままBレジスタ、Cレジスタとして使用されます。条件分岐はBレジスタ、Cレジスタとも、それぞれ1ヵ所ずつありますが、フローチャートの書式が微妙に違っています。これは、なにも書き間違えたわけではなく、BレジスタにはBASICのFOR～NEXTに当たるような特殊な命令があるからなのです。くわしくはニモニックの命令説明の時にするとして、早速プログラムの入力、セーブ、実行に移ることにしましょう。

なお、この**LIST3-4**からは、リスト中にもコメントを入れてありますので、本文中にあるはじめての命令に対する説明とあわせて読んでください。

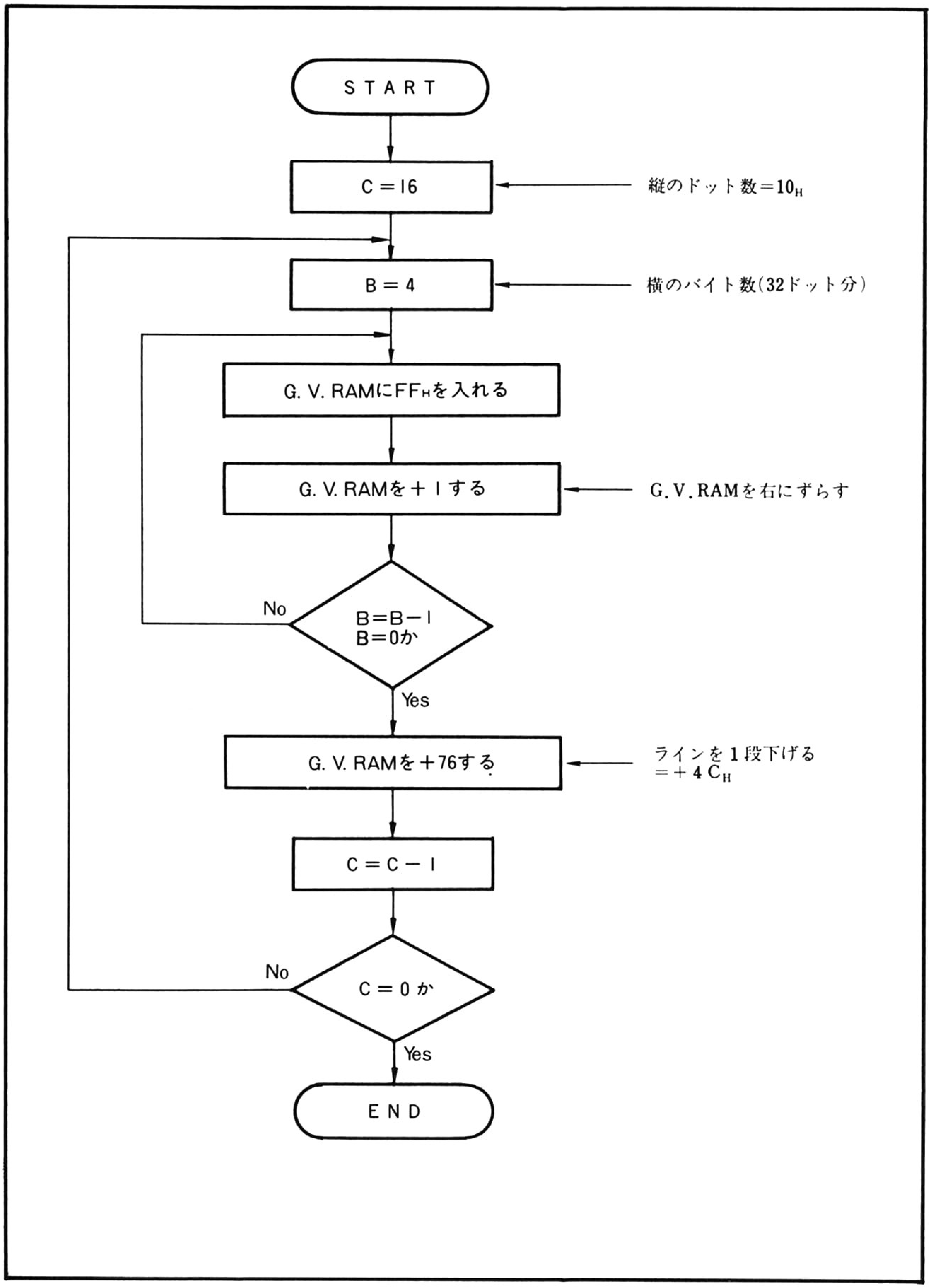


図3.5　32×16ドットの正方形表示のフローチャート

LIST 3-4

BD00	F3	DI		割込み禁止。
BD01	21 C000	LXI	H,C000	HL←C000$_H$……G.V.RAMアドレス。
BD04	11 004C	LXI	D,004C	DE←004C$_H$……右エンドから次段への増加バイト数
BD07	3E FF	MVI	A,FF	A←FF$_H$
BD09	D3 5C	OUT	5C	ブルー面にバンク切り換え。
BD0B	0E 10	MVI	C,10	C←10$_H$……縦のライン数。
BD0D	06 04	MVI	B,04	B←4………横のバイト数。
BD0F	77	MOV	M,A	HLで示されるアドレスにAの値を入れる。
BD10	23	INX	H	HL←HL+1
BD11	10 FC	DJNZ	BD0F	B←B−1をし、B≠0ならばBD0F$_H$番地へジャンプ。
BD13	19	DAD	D	HL←HL+DE……HLを次ラインの先頭アドレスにする。
BD14	0D	DCR	C	C←C−1
BD15	20 F6	JRNZ	BD0D	C≠0ならばBD0D$_H$番地へジャンプ。
BD17	D3 5F	OUT	5F	メインRAMにバンク切換え。
BD19	FB	EI		割込みの許可。
BD1A	FF	RST	7	h]の状態へ戻る。

実行の結果は、画面に表示された通りです。

■〔応用問題〕正方形の位置を変える

さて、つぎに、この正方形の表示場所を変えようというわけですが、それにはプログラムのどこを変更すればよいのでしょうか？　任意の位置にこの青い正方形の表示をさせるにはどうすればよいのでしょうか？　すでにおわかりの方もいるかもしれませんが、その問題を考える前に、このプログラムではじめて出てきたニモニックについて、まず理解をしてください。

ニモニック （読み方）	説　　　明
LXI H, C000 （ロードエックス イミーディエート）	HL レジスタに $C000_H$ を代入する。細かくいえば、H レジスタに $C0_H$ が、L レジスタに 00_H が入る。LXI の L は Load の略で、これは Move と同じ意味に使われる。X は、このようにニモニックの頭にきていない時はすべてペアレジスタを表す記号となる。I は MVI の時の I と同じく、Immediate の略で即値を示す。
MOV M, A （ムーブ）	HL レジスタで示されるアドレスにアキュムレータの値を入れる。MOV は MOVe の略だが、第1オペランドにある M は HL レジスタで指し示されるアドレスの中身を意味する記号。この場合、HL レジスタの値が $C000_H$ なので、$C000_H$ 番地にアキュムレータの値を入れることになる。
INX H （インクリメント　エックス）	HL レジスタの値を＋1する。INX とは INcrement X（ペアレジスタ）の略である。ちなみに－1する場合は DCX（DeCrement X（デクリメントエックス）の略）となる。
DJNZ BD0F （デクリメント ジャンプ イフノンゼロ）	B＝B－1をし、B<>0なら $BD0F_H$ 番地へジャンプする。BASIC の FOR～NEXT 文に似た命令であるが、ジャンプできるのは現在のアドレスを基準にして、-80_H～$+7F_H$ のあいだに限られる。DJNZ とは、Decrement Jump if Non Zero の略。
DAD D （ダブル アド）	HL レジスタに DE レジスタの値を加える。DAD は Double ADd の略であるが、加算できるのは HL レジスタと第1オペランドで示されるペアレジスタとに限られる。

DCR C (デクリメント)	Cレジスタの値を −1する。DCRとはDeCRement の略で、オペランドにあるレジスタの値を −1する。反対に +1する場合はINR(INcRement〔インクリメント〕の略) という。前述のペアレジスタの時との違いに注意。
JRNZ BD0D (ジャンプ リラティブ イフ ノンゼロ)	ここでは、C<>0ならばBD0D$_H$番地へジャンプするという意味。JRNZはJump Relative if Non Zeroの略であるが、Relative JumpはDJNZでも使われているが**相対ジャンプ**のことで、現在のアドレスを基準にして−80$_H$〜+7F$_H$のあいだにしかジャンプできない。 これに対して、**絶対ジャンプ**（ニモニックにより JMP または J と略される）というどこでも指定のアドレスへジャンプできる命令もある。 両者は、実行速度と命令に必要なメモリ数（相対ジャンプ＝2バイト、絶対ジャンプ＝3バイト）、および分岐できる条件に若干の違いがある。

■2バイトの数値は逆になる

プログラムには条件分岐によるループが入り、だんだんとマシン語プログラムらしくなってきました。さて、さきほどの質問の答えですが、HLレジスタの値を変えれば画面のアチコチに青い正方形を描くことができるはずです。

その場合、Aコマンドでアドレス変更をしていると気がつかないかもしれませんが、EコマンドまたはSコマンドによってBD02$_H$、BD03$_H$番地の中を直接変更しようとすると、C000ではなく00 C0と数値が逆に入っているのが目につきます。これはバグではなく、2バイト単位の数値はメモリには上位・下位が逆になって入るようにCPU自身が設計されているためです。

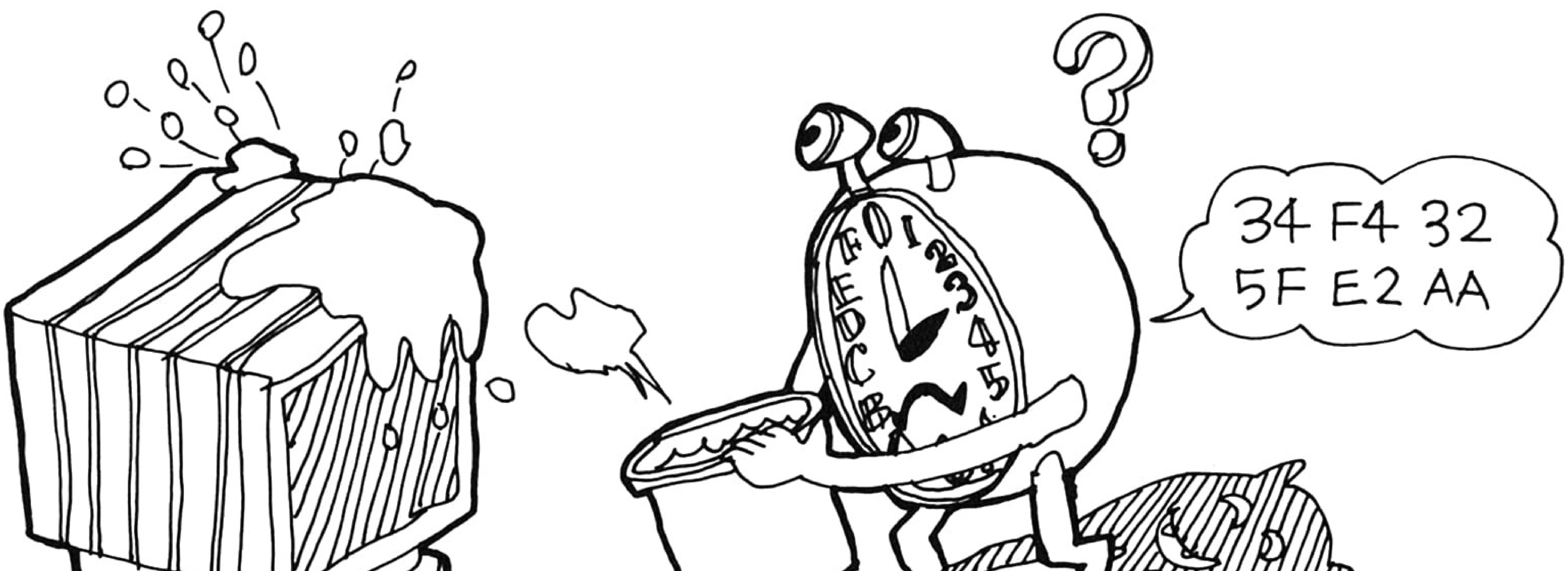
たとえば、ここではまだ出てきていませんが、ペアレジスタの値を一度にメモリに入れる場合なども、同じように下位・上位の順でストアされるので

す。しかし、メモリからペアレジスタに読み出す時にも、下位・上位の順でレジスタに入りますから、通常のプログラムにおいては全く気にすることはありません。

■〔演習問題〕

マシン語の練習のためには、この程度のやさしいプログラムを少しずつ変更してみることが意外に役に立ちます。

そこで宿題ですが、このプログラムの一部を変更して、グラフィック画面全部を青でうめてみてください。解答はどこにもありませんから、あくまでも自分の力でやらなければなりません。もしも、それができないようだと、先へ進むことは自信喪失につながりますから、もう一度わかる所までもどったほうが賢明です。たぶん、本書の読者の中にそのような方はいないと思いますが……。



3-4 フラグの存在

いま、この文を読まれている方は、前の節でふれた条件分岐を思いのままに使って、オールマシン語で画面を全部青にするプログラムができる実力を備えたエリートばかり、というはずでしたね。そこで、ここでは休憩の意味も込めて、プログラムから少しばかり離れることにします。

これは、つまり強制的な分岐になるわけですが、では何をやるかというと条件分岐についてなのです。何をいっているのか、わけのわからない方……、もう一度「先頭から読み直し」へ条件分岐です!!

■フラグレジスタの役割

前節におけるニモニック（JRNZ BD0D）の説明で「ここでは、C＜＞0ならば……」と奥歯にモノがはさまったような、不透明な表現があったのを思いだしてください。なぜ、このような表現をしなければならなかったのでしょうか。

それは、ここでの条件分岐はCレジスタの値など全く見ていない、という恐しい事実があったからなのです。ただ、結果的にあのプログラムに関しては、Cレジスタの値を見て判断したことと変わりがないことをしました。だからこそ正常に動いたわけですが、実際は何を見て判断したのか、そして、なぜ、あのプログラムで正しかったのか、これはハッキリさせなければならない問題なのです。

まず第1の疑問に対してですが、見たのは**フラグレジスタ**の中にある**ゼロフラグ**というのが正しい答えです。

アッサリ答えを書いてしまったのは、この答えの意味を理解するほうがもっと大変だからです。

このフラグレジスタというのは、足し算・引き算や比較など各種の演算を

行った結果によって、ある決まった反応を示す特殊なレジスタです。当然、数値を代入するような使い方は、これまでに出てきたレジスタと違い、できません。そして図3.6のように、ビット単位で使い分けされるのです。

ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
Sフラグ	Zフラグ	未使用	Hフラグ	未使用	P/Vフラグ	Nフラグ	Cフラグ

図3.6　フラグレジスタの内容

個々のフラグの役割を表にまとめてみましょう。

フラグ名	正式名称	内容
Sフラグ	サインフラグ	演算結果のビット7の値がそのまま入る.
Zフラグ	ゼロフラグ	演算結果がゼロなら1，ゼロでなければ0となる
Hフラグ	ハーフキャリーフラグ	演算の結果，ビット3と4の間で移動があれば1，なければ0となる.
P/Vフラグ	パリティフラグ	論理演算の結果，1の立っているビットの総数が偶数なら1，奇数なら0となる.
	オーバーフローフラグ	算術演算の結果，補数表示で正しい答にならない時，1となる.
Nフラグ	減算フラグ	減算・比較命令の後は1，それ以外は0となる.
Cフラグ	キャリーフラグ	1バイト同士の加算では，その結果がFF_Hを越えると1，FF_H以下なら0となる. 2バイト同士の加算では，その結果が$FFFF_H$を越えると1，$FFFF_H$以下なら0となる. 減算／比較では，引く数または比効する数の方が，元になる数より大きければ1,同じまたは小さい時には0となる.

フラグレジスタの役目とは、このように演算結果に対して、ビットを1にしたり0にしたりして反応することです。

このフラグが1になることを「フラグが立つ」といいます。この言いまわしになれることは大切で、とくにゼロフラグにおいては「ゼロになったら1になる」などとおぼえようとすると混乱しますから、「ゼロになったらゼロフラグが立つ」とおぼえるほうが賢明といえましょう。

そうはいっても、いきなりこの反応のしかたを全部おぼえることはありま

せん。よく使用されるのはゼロフラグとキャリーフラグの2つだけで、残りはマシン語に相当自信がついてからで十分です。というより、ゼロフラグとキャリーフラグさえ使いこなせれば、まずこまることはないといえるほどです。また、何ビットめが何フラグという関係を知ることは全く不要ですから、永久におぼえる必要はありません。

■反応が異なることがある

フラグレジスタによる条件分岐で面倒なのは、じつはこれからなのです。その理由は、ニモニックによっては、同じような内容の命令でも、反応のある場合とない場合があるからです。

たとえば、DAD命令では、加算の結果がゼロになってもゼロフラグの変化はありません。また、ニモニックの中にはフラグレジスタが全く反応を示さない命令もあります。したがって、自分では条件分岐させたつもりでも、フラグレジスタに反応がなければ、条件分岐にならなくなってしまう危険性があるわけです。

ここで間違わないためには、ただひたすら、なれとインストラクション表によるニモニックの確認しか方法はありません。本書のプログラムを入力するだけであれば問題はないのですが、いずれはオリジナルなプログラムを組むわけですから、そういった細かな点にも留意するようにしてください。

■第2の疑問に答える

さて、第2の疑問点についてですが、これの答えは今の説明の中にヒントが入っているのです。

もう一度プログラムを見てみましょう。行番号ふうにいうなら、アドレスBD14ではDCR　Cによってゼロフラグが反応している。つぎのBD15ではJRNZで条件分岐させている。つまりこの2つの命令のあいだにはゼロフラグが反応するような命令が入っていないから、誤動作は起きなかったというわけです。

わかりにくいかもしれませんから、逆接的につぎのようなケースを想定してみましょう。

```
DCR   C
MVI   C, 00
JRNZ  xxxx
```

このプログラムでは条件分岐させる前にC=0としていますから、もしもCレジスタの値を見ているならば、絶対にジャンプしないはずです。また、MVI C, 00によってゼロフラグが立つならば、これも絶対にジャンプすることはないはずです。

ところが、実際にはDCR Cを実行した結果、C<>0ならば、このプログラムはキチンとxxxxへ条件分岐が行われるのです。

この理由は、MVI C, 00という命令では、フラグレジスタは何の反応も起こさないからです。そのため、この条件分岐はDCR Cの結果によるフラグの変化、すなわちDCR Cの時点でC<>0ならばジャンプをする、ということになってしまうわけです。

マシン語になれてくると、この例のように、フラグレジスタの反応と条件分岐とのあいだに別な命令を入れるということも少なくはありません。しかし、現在のところは「フラグレジスタの反応と条件分岐は常に1セット」と考えてプログラムを組むほうが確実でいいかもしれませんね。

ここで巻末のインストラクション表にあるフラグの見方、つまり記号の意味を示しておきすので、参考にしてください。

記号	意味
·	フラグの変化はない
1	必ずフラグが立つ
0	必ずフラグがリセットされる
↕	演算の結果により，フラグが変化する

図3.7 フラグレジスタ表の見方

3-5 白い正方形

青い四角形の表示に関してはすでに自由自在といった気分かもしれませんが、今度は白で例の32×16ドットの正方形を描いてみましょう。カラー発色の原理がわかっていれば、理論的には簡単なことです。**LIST3-4**にあるOUT 5Cの部分を、5Dと5Eについても実行すればいいだけの話です。

しかし、これまでの知識だけでプログラム化しようとすると、正方形を描くルーチンを3面分、長々と書かなければならず、けっこう面倒なことになります。これが、それぞれ全く違ったことをするのであれば仕方がないのですが、内容が同じ正方形を描くというだけになおさらです。

■サブルーチン

BASICであれば、このような時には、当然、GOSUB～RETURN文でサブルーチン化してしまうはずです。といっても、それは内容がもっと複雑な時で、この程度ならその必要もないといえますが、マシン語ではよほど速度に重点を置いた時以外は、共通部分はどんどんサブルーチンにしていきます。

これは、メモリの有効利用のほかに、プログラムに際しての入力ミス防止や、デバッグ作業の省力化につながるからです。

本格的なマシン語プログラムの場合は、一度の入力でパーフェクトに動くことはまず考えられません。そこで、デバッグという手直し作業を何度も何度も行うわけですが、その時にプログラムがダラダラしていると非常にわかりにくいものなのです。これがサブルーチン化してあれば、ルーチン単位でチェックできるので、バグを発見しやすくなるというわけです。

マシン語におけるサブルーチンの用法は原理的にはBASICと全く同じですが、こちらはCALL～RETという命令によって行います。この命令を正方形を描くルーチンに用いたのが、これから作成する**LIST3-5**になるわけで

LIST 3-5

```
BD00   F3          DI            割込み禁止。
BD01   21 C000     LXI  H,C000   HL←C000H.
BD04   11 004C     LXI  D,004C   DE←004CH.
BD07   3E FF       MVI  A,FF     A←FFH.
BD09   D3 5C       OUT  5C       ブルー面にバンク切り換え。
BD0B   CD BD1C     CALL BD1C     正方形表示ルーチンをコール。
BD0E   D3 5D       OUT  5D       レッド面にバンク切り換え。
BD10   CD BD1C     CALL BD1C     正方形表示ルーチンをコール。
BD13   D3 5E       OUT  5E       グリーン面にバンク切り換え。
BD15   CD BD1C     CALL BD1C     正方形表示ルーチンをコール。
BD18   D3 5F       OUT  5F       メインRAMにバンク切り換え。
BD1A   FB          EI            割込みの許可。
BD1B   FF          RST  7        元の状態へ戻る。
BD1C   E5          PUSH H        HLの値をスタックへ退避。
BD1D   0E 10       MVI  C,10     C←10H……縦のライン数。
BD1F   06 04       MVI  B,04     B←4…………横のバイト数。
BD21   77          MOV  M,A      HLで示されるアドレスにAの値を入れる。
BD22   23          INX  H        HL←HL+1。
BD23   10 FC       DJNZ BD21     B←B－1をし、B≠0ならばBD21H番地へジ
                                 ャンプ。
BD25   19          DAD  D        HL←HL+DE……HLを次ラインの先頭アド
                                 レスにする。
BD26   0D          DCR  C        C←C－1
BD27   20 F6       JRNZ BD1F     C≠0ならばBD1FH番地へジャンプ。
BD29   E1          POP  H        HLの値をスタックから取り出す。
BD2A   C9          RET           リターン。
```

す。

この中で$BD1C_H$～$BD2A_H$番地が正方形を描くためのサブルーチンで、グラフィックV.RAMをバンク切り換えするごとにコール、つまり呼び出されています。

■PUSH と POP

サブルーチンの先頭と最後に、PUSH H と POP H という初めて登場するニモニックが入っています。これは、正方形を1面描くと HL レジスタで示されるグラフィック V.RAM のアドレスが変化してしまうため、ルーチンの最初で一時的に HL レジスタの値を別な場所に退避（記憶）しておき、最後にそれを取り出してもとにもどすということをしているのです。

この考え方は、変数を自由に作れる BASIC にはないものですが、レジスタをヤリクリしなければならないマシン語プログラムにおいては大変便利なものとなっています。ただ便利であるといっても、そこにはつぎのような非常にシビアな使用上の制限がありますから、注意が必要です。

まず、一時的に記憶する場所についてですが、コンピュータの中である以上、これはやはりメモリしかありません。では、メモリのどこにそのようなエリアがあるかというと、モニタに入る前に実行した CLEAR 文により保証されたマシン語エリアの範囲外に16バイト確保されるものです（番地の若いほう）。

PUSH/POP という命令は、ペアレジスタを単位として扱いますから、1回の実行につきメモリが2バイト必要になります。したがって16バイトあるということは、8レベルまでの使用がOKということです。

■スタックエリアとスタックポインタ

ところで、これまで出てきたレジスタのうち、Aレジスタとフラグレジスタはそれぞれ単独のレジスタでしたから、このままでは便利な PUSH/POP 命令が使えません。それでは不便だというので、使う時には孤独な独身同士でペアを組み、PSW（Program Status Word の略）となってペアレジスタに化けてしまうことにしてあります。

この16バイトの避難場所のことを**スタックエリア**といい、ここに記憶されたデータのアドレスを管理している人物（？）を**スタックポインタ**といいます。

PUSH命令によってスタックエリアに送られたデータは次つぎと入って

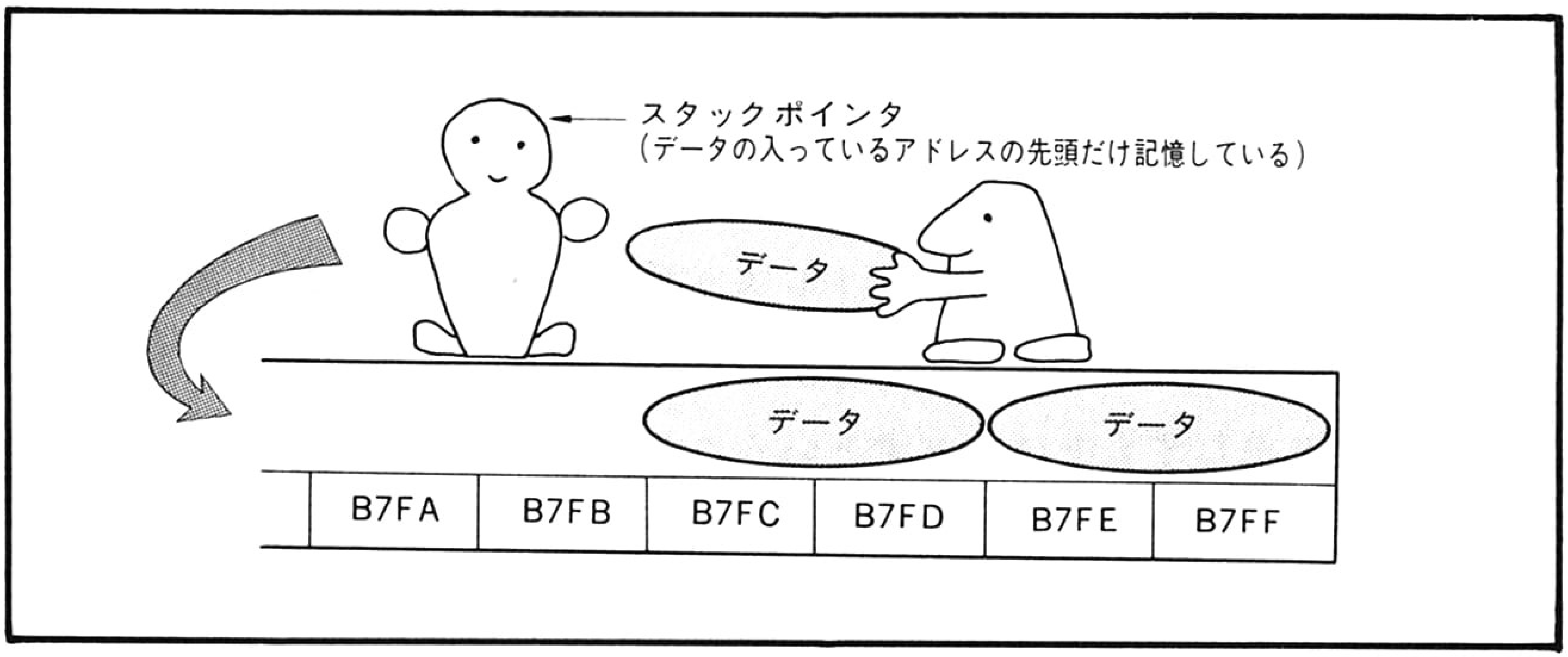


図3.8　スタックエリアとスタックポインタ

いって記憶されますが、ふたたびデータとして取り出す時には、最大で8つおさまっているうちのどれでも好きなものを取り出すというわけにはいきません。いちばん最後に入れたものから順に出さなければならないのです。

■あと入れ、先出し

スタック (stack) とは、もともと麦わらの束などを畑に積み上げることをいうのですが、これだと、いちばん上から取らなければ当然、山がくずれてしまいますよね。このような関係を、“Last In Fist Out” または “First In Last Out”の原則といいます。「あと入れ、さき出し」というわけです。たとえば、現在のスタックポインタを $B800_H$番地とすると、データの出し入れは図3.8のようになります。

最初にスタックへ PUSH されるデータは $B7FE_H$, $B7FF_H$番地に入り (つまり 2 バイト)、その時のスタックポインタは $B7FE_H$番地となります。2 番目のデータは $B7FC_H$, $B7FD_H$番地に入り、スタックポインタは $B7FC_H$になります。このようにスタックポインタは、データを取りにきた時にすぐ出せるアドレスになっているのです。これが、記憶した逆の順にしかデータを出せない理由です。

また、スタックポインタがおぼえているのはアドレスだけですから、出し入れをしたレジスタの名前はいっさい気にしていません。だから、PUSH したのが HL レジスタからでも、POP したのが DE レジスタであれば、その

データはアッサリとDEレジスタに入ってしまいます。これは、テクニックとして意識的にそうすることもあるのですが、なれるまでの辛抱が必要です。

■覚えているのは番地だけ

これだけの注意事項であれば、そう面倒なことではありませんが、このスタックポインタとスタックエリアというのはCALL～RET命令のもどりアドレスを記憶することにも使用されるので、ことはちょっと複雑です。

というのも、いくらコンピュータでも、プログラムの実行中にRET命令に出会っただけでは、どのアドレスへもどっていいのか知りようがありません。そのため、CALL命令があったら、必ずそのもどり番地を記録しておかなければならないわけです。たとえ、このような重要なアドレスであっても、管理しているのはスタックポインタですから、ほかのレジスタが間違って取りにくれば素直に渡してしまうのは変わりありません。そうなれば、プログラムは暴走、画面はメチャクチャ、時にはリセットまでかかり、大変なことになってしまいます。

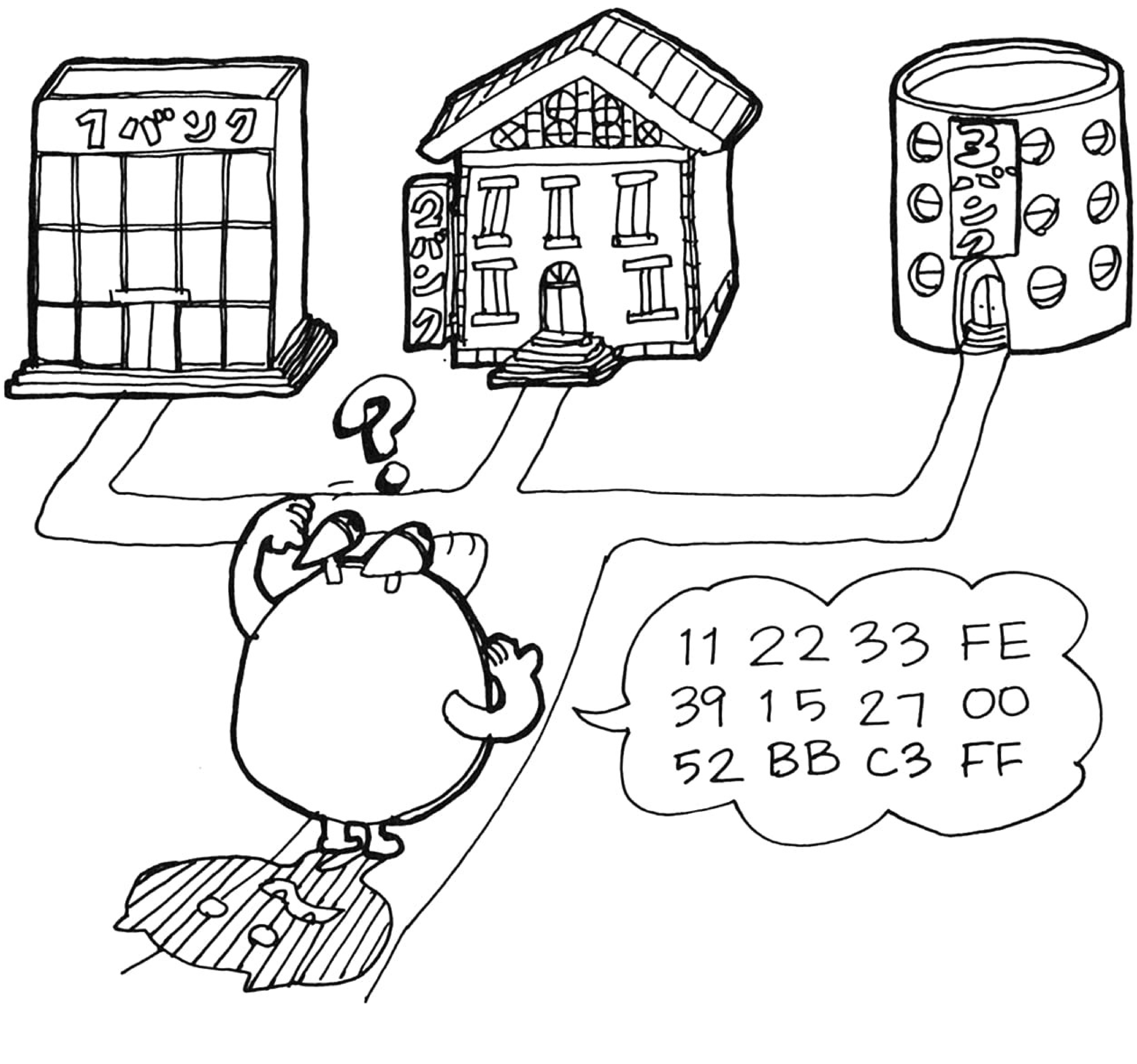
しかし、このスタックポインタのことを監督できる偉い人物もキチンと控えていますので、安心です。その人物とは、もちろんプログラムを作るあなたのことです。つまり、プログラムを組む際にスタックポインタが狂わないように、あなたが気配りをすればいいだけのことです。基本的には、1つのルーチンの中でPUSH/POPの使用回数が同じであれば、それほど気にすることはないでしょう。

■スタックエリアを拡張するには

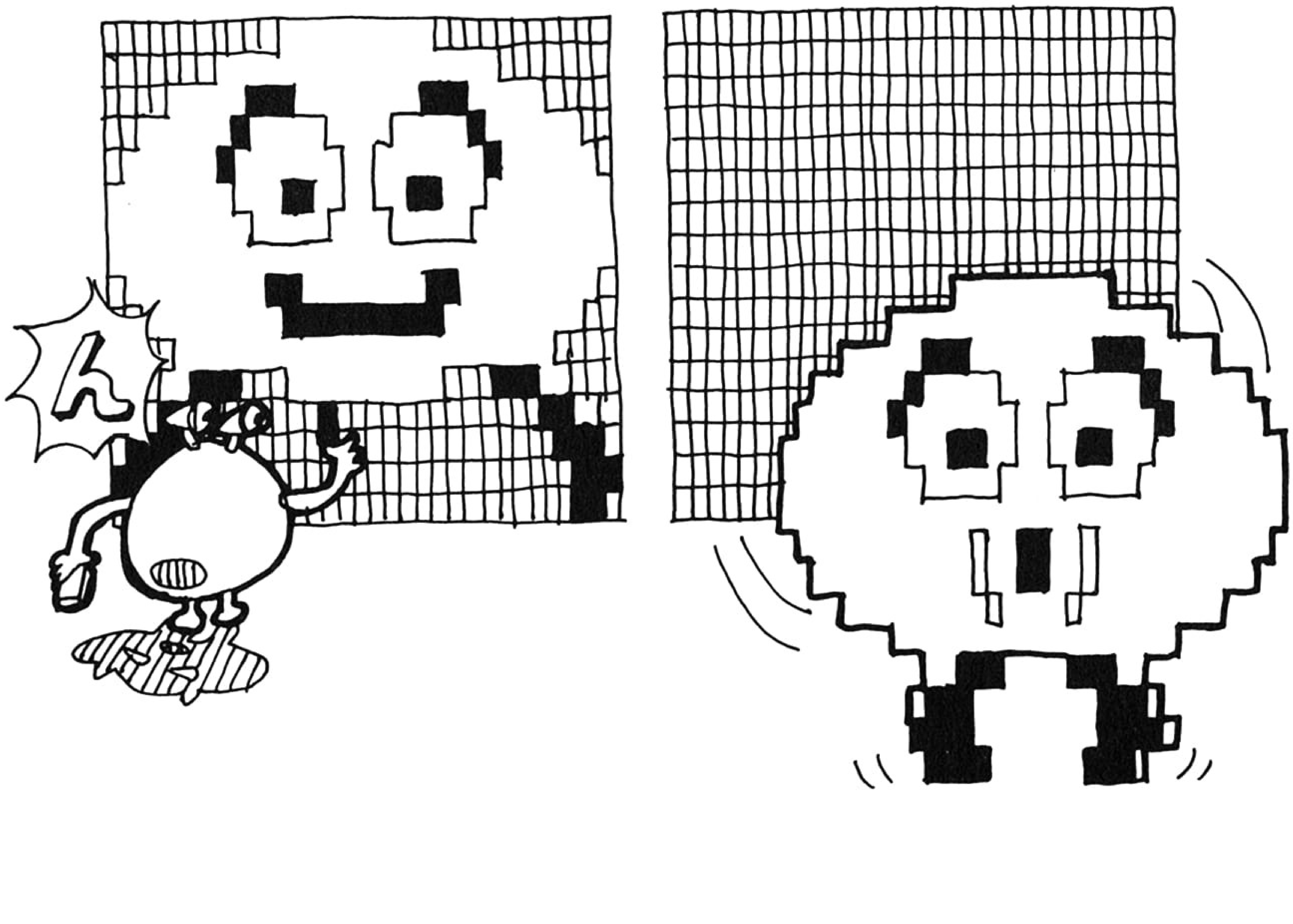
なお、スタックエリアが16バイトでは足りないような場合には、プログラムを組む前にマシン語エリア内にスタックエリアを確保しなければなりません。これは、LXI SP, xxxxとすれば、スタックポインタを好きな所に設定できますから簡単です。ただし、BASICにもどるにはモニタに入った時のスタックポインタになっていなければなりませんから、BASICのサブルーチンとして使う場合は、できるだけ16バイト（8レベル）以下にしておいたほうが無難です。

■番地の若いほうに設定すること

もう１つの注意点は、スタックエリアはバンク切り換え中にも当然使用されますから、C000$_H$番地より番地の若いほうへ設定していないと、バンク切り換えと同時にデータも切り離されてしまうということです。もちろん、バンク切り換えそのものもC000$_H$番地以前で行わないと、プログラム自体までが切り離されてしまい、どちらも暴走という悲惨な結果になってしまいます。



第4章
キャラクターパターンの出現



最近はパソコンゲームのレベルもますます高くなってきており、以前のようにストーリーを考えることから音響効果に至るまで、何から何までひとりで作り上げるというケースはあまり見かけられなくなってきました。これは、制作する側の都合も少しはあると思いますが、やはりユーザーの本物志向がそういう流れにさせたと見るべきではないでしょうか。

グループで作るといっても、単に分業にしただけではなく、シナリオライター、イラストレーターや漫画家、ミュージシャン、そしてプログラマーと、それぞれが専門の分野のことを担当するわけですから、内容がよくなるのも当然の結果であるといえます。

こうなると、もはやBASICだけでできているゲームなど、とうてい考えられない状況です。とくに、グラフィック表示に関してはマシン語が必需品といってもいいほど、スピードが要求される時代になってきています。画面を動きまわる小さなキャラクターパターン作りは、まさにゲーム・グラフィックスへの第1歩かと思います。

4-1 座標とアドレス

青い正方形から白い正方形へとプログラムは進化してきましたが、どちらも練習用プログラムに変わりはありませんでした。これから作っていくキャラクターパターン表示プログラムにしても、しょせんは本書の中だけのメインプログラムですから、客観的に見ればやはり練習用プログラムかもしれません。

しかし、何はともあれわれわれのマシン語プログラミングの知識を実際にゲームに使えるだけのレベルにしようということで、本書の最後の章は、標準的なパターン表示ルーチンを仕上げることが目標です。といっても、座標(X，Y）にパターン番号で示されるパターンを表示するというだけのことですが、これまでのように手作業でプログラムを変更するのではなく、ワークエリアを持つ本格的（？）なものへの挑戦です。

■座標のとり方

まず座標についてですが、これまでの知識から、横は1バイト、すなわち8ドット単位がいちばん都合がいいことがわかります。縦は、とくにそのような制限をつける必要はないのですが、横と同じサイズのほうがいろいろと便利なので、4ドットを1単位とするのが普通です。

したがって、640×200ドットのグラフィック座標が、80×50で表現されることになるわけです。これはリアルタイムゲームでは標準的な座標であり、キャラクターパターンの動きの単位でもあります。

このような座標体系とグラフィックV.RAMとの関係を、計算式で表すとつぎのようになります。

グラフィックV.RAMアドレス＝$C000_H+140_H\times Y+X$

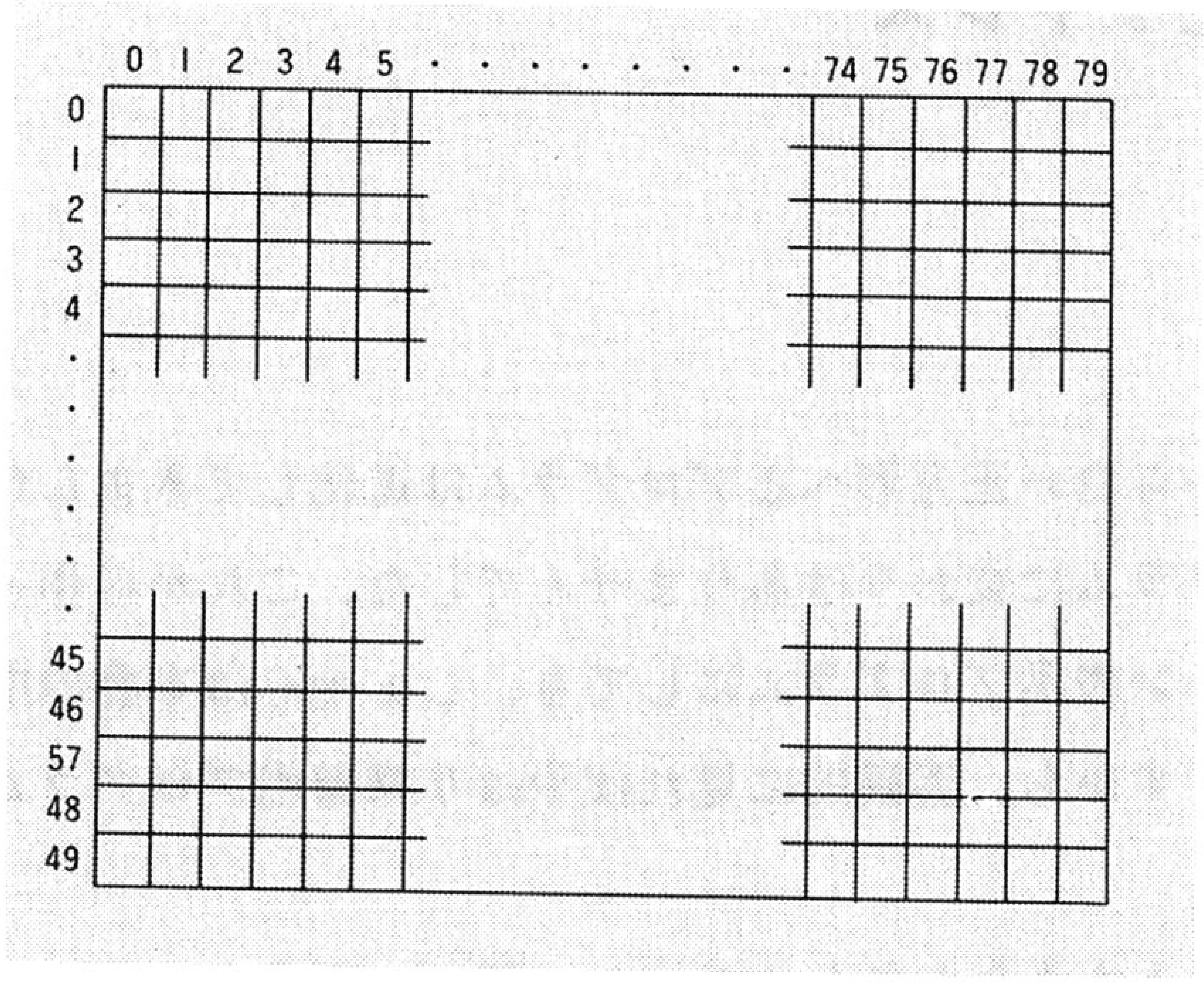


図4.1　ゲーム用標準座標

140_Hは十進数では320（80×4）になり、縦4ライン分のアドレス変化量です。簡単な式ですが、マシン語にはかけ算の命令がないため、このかけ算は足し算を繰り返して処理しなければなりません。$140_H + 140_H$…（Y回繰り返す）…$+140_H$とがんばるわけです。

■ループ――BASICとのちがい

繰り返しにはおなじみのループを使いますが、Y=0になる可能性もあるので、このままプログラムしてはいけません。ループ処理（140_Hを加算すること）に入る前に、Y=0か否かをチェックするプログラム（Y=0ならばループ処理をスキップする）を入れないといけないのです。

これは、BASICの繰り返し命令であるFOR～NEXT文との大きな違いで、たとえDJNZを使ったとしても、何もしなければ1回はループ処理を通ってしまうことになるからです。それどころか、1バイトの数値では0－1＝FF_Hとなるため（図2.2参照）、実際には256（100_H）回もループ処理されることになってしまいます。

そこで、これらの不都合を避けるため、この計算式をつぎのように変形します。

グラフィックV.RAMアドレス＝$BEC0_H$＋140_H×（Y＋1）＋X

これで、Y＝0の時でも1回のループ処理がされることになり、プログラムも組みやすくなったわけです。

■本格的訓練開始

なお、このプログラムからはつぎのプログラムをアペンドしながら、少しずつ成長させていきます。そのため、プログラムもメイン部とサブ部とに分け、メインがその都度成長するプログラム、サブは必要に応じてメインを実行確認するためのテストプログラムとなっています。

またプログラムとは別に、表示座標やパターン番号を示すワークエリアも設定してあります。それぞれの使用予定アドレスは、つぎのようになっています。

パターンデータ	：$B800_H$～	（第3節で作成）
メインプログラム	：$BD00_H$～	（第1節、第2節で作成）
ワークエリア	：$C500_H$～	（第4節で利用）
サブプログラム	：$D000_H$～	（第2節より作成）

これから作る〈座標→アドレス変換プログラム〉はメインの先頭にくるもので、レジスタ名で表現すると座標（C，B）にパターン番号Aを表示する、というプログラムの一部に当たります（**LIST4-1**）。

■XCHG

このリストではじめて見るニモニックに、XCHG（eXCHanGeの略）というのがありますが、これはHLレジスタとDEレジスタの値を交換するという命令です。この命令により交換できるのは、あくまでもHLレジスタとDEレジスタの内容だけで、それ以外のレジスタには使用できません。ここでは、計算によりHLレジスタに求められた表示アドレスを、DEレジスタに移すために用いています。

LIST 4-1

```
BD00   21 BEC0    LXI  H,BEC0   HL←BEC0H.
BD03   11 0140    LXI  D,0140   DE←0140H.
BD06   04         INR  B        B←B+1.
BD07   19         DAD  D        HL←HL+DE.
BD08   10 FD      DJNZ BD07     B←B－1し、B≠0ならば，BD07H番地ジャンプ。
BD0A   09         DAD  B        HL←HL+BC……BはDJNZで0になっている。
BD0B   EB         XCHG          HL↔DE……DE＝表示アドレスとなる。
```

今回はサブとしてのプログラムはなく、メインをダイレクトに走らせてテストの実行をすることになります。しかし、このプログラムはこれまでと違って中途半端で終わっているうえ、BCレジスタの値が特定されていないので、このまま実行するというわけにはいきません。BCレジスタの値は、本来ならばサブのほうでワークエリアから取り出して設定されるべきなのですが、まだメインが未完成のため時期尚早です。

■Xコマンド

そこで、X（イグザミン）というコマンドにより、まずBCレジスタの値を設定します。Xコマンドは、単にh]X ⏎ とした時は全レジスタの値を表示するだけですが、Xのあとにレジスタ名を入れると、そのレジスタの値を変更できるようになります。

変更前のBCレジスタ＝0000Hの場合

```
h]XB⏎
B:0000=▮
```

この状態で、カーソルの位置から変更したい数値を入力すればいいのです。練習ですから、アドレスがわかっている適当な座標（C，B）を入れてみてください。スペースバーを押せば、連続して各レジスタの値を変更することができます。終わる時はリターンキーでh］にもどります。

■**ブレークポインタ**

つぎに実行ですが、これは未完成のプログラムですから**ブレークポインタ**を設けて途中でストップさせなければなりません。

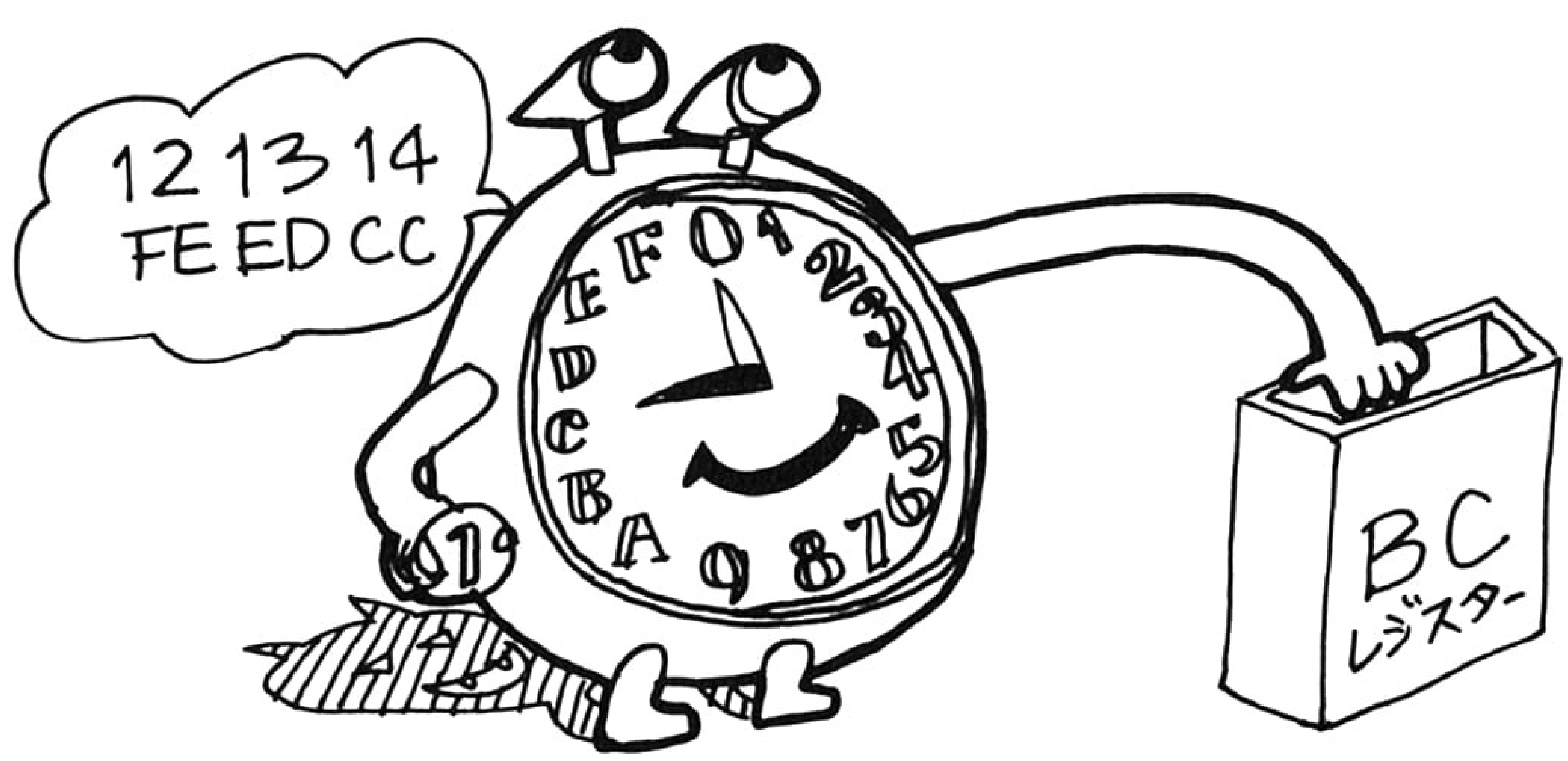
マシン語プログラムをブレークさせるには、実行命令であるGコマンドの開始アドレスに続けて、終了アドレスのつぎのアドレスをカンマで区切って入力するだけです。ただし、ブレークが可能なのは、メインRAM上に限られます。

それでは実際にやってみましょう。

```
h]GBD00, BD0C⏎
h]▮
```

これで、$BD00_H$番地から$BD0B_H$番地までのプログラムが実行されたわけですが、結果はDEレジスタに入っているはずです。Xコマンドでレジスタの値を表示させてみて、DEレジスタの値が予定通りのアドレスになっていれば正しく計算されているということです。

Xコマンドは、これから先非常に重宝になるものですから、いろいろとBCレジスタの値を変えて1つひとつ確認してみてください。



4-2 パターン表示ルーチン

一口にリアルタイムゲームといっても、その内容により表示されるパターンのサイズは全く違っています。もちろん、同じゲームの中にあっても、パターンサイズを統一する必要などまるでないのですが、たとえば迷路型のゲームであれば、迷路からはみ出さないサイズというのは最低の条件として存在するわけです。

その程度の判断レベルにおいて、一般的なパターンサイズといえば、これまで青や白で何度も描いてきた例の正方形（32×16ドット）が、まさに標準サイズであるといえます。そこで、本書ではパターン番号によるパターンの区別はつけるものの、そのサイズは、すべてこの32×16ドットということにしています。

■パターン表示の実際

では、この正方形をどのようにすればパターンが表示できるようになるのでしょうか。すでにいろいろな実験をされてわかっていると思いますが、これまでグラフィックV.RAMに入れていたFF_Hを、パターン表示用のデータに置き変えればいいわけです。具体的なパターンデータを示すのは後回しにすることにして、データの総数をまず数えてみましょう。

図3.4を見ると、ハッキリしますが、1面当たりのデータ数は64必要です。したがって3面では、

$$64\times 3=192$$
$$=C0_H$$

となります。パターンデータは$B800_H$番地から入れることにしていますので、パターン番号別のデータ内訳は図4.2のようになります。ただし、本書で

パターン番号	ブルー面のデータ	レッド面のデータ	グリーン面のデータ
0	$B800_H$ ～ $B83F_H$	$B840_H$ ～ $B87F_H$	$B880_H$ ～ $B8BF_H$
1	$B8C0_H$ ～ $B8FF_H$	$B900_H$ ～ $B93F_H$	$B940_H$ ～ $B97F_H$
2	$B980_H$ ～ $B9BF_H$	$B9C0_H$ ～ $B9FF_H$	$BA00_H$ ～ $BA3F_H$
3	$BA40_H$ ～ $BA7F_H$	$BA80_H$ ～ $BABF_H$	$BAC0_H$ ～ $BAFF_H$
4	$BB00_H$ ～ $BB3F_H$	$BB40_H$ ～ $BB7F_H$	$BB80_H$ ～ $BBBF_H$
5	$BBC0_H$ ～ $BBFF_H$	$BC00_H$ ～ $BC3F_H$	$BC40_H$ ～ $BC7F_H$

図4.2　パターン・データの内訳

は $BD00_H$番地からはプログラムエリアとしたため、現状で扱えるパターン数はこの6パターンが限度です。

■データアドレスの計算

さて、パターンを表示するアドレスは、すでにDEレジスタに求められているわけですが、今度はアキュムレータで表されているパターン番号から、データアドレスを求めなければなりません。この計算式は、つぎのようになります。

$$\text{パターンデータ・アドレス} = B800_H + C0_H \times A$$
$$= B740_H + C0_H \times (A + 1)$$

式を変形した理由は、もうおわかりですね。これをプログラム化したのが**LIST4-2**で、これが**LIST4-1**に続くのはいうまでもないことです。

■入力データの確認テスト

入力したらアキュムレータの値を適当なパターン番号にしてから、つぎのようにテストをしてみてください。

```
h]GBD0C, BD17⏎
h]X⏎
```

LIST 4-2

```
BD0C    21 B740     LXI  H,B740    HL←B740H.
BD0F    01 00C0     LXI  B,00C0    BC←00C0H……1パターンのデータ数。
BD12    3C          INR  A         A←A+1.
BD13    09          DAD  B         HL←HL+BC.
BD14    3D          DCR  A         A←A−1.
BD15    20 FC       JRNZ BD13      A≠0ならBD13H番地へジャンプ。
                                   (HL=パターンデータ・アドレスとなる)
```

HLレジスタの値が、図4.2と比べて間違いがなければOKです。

これに続けて、パターン表示ルーチンの残り**LIST4-3**を入力してください。同時に、D000$_H$番地からのサブプログラム(**LIST4-4**)も入力します。

LIST 4-3

```
BD17    D3 5C       OUT  5C        ブルー面にバンク切り換え。
BD19    CD BD29     CALL BD29      正方形をデータで描く。
BD1C    D3 5D       OUT  5D        レッド面にバンク切り換え。
BD1E    CD BD29     CALL BD29      正方形をデータで描く。
BD21    D3 5E       OUT  5E        グリーン面にバンク切り換え。
BD23    CD BD29     CALL BD29      正方形をデータで描く。
BD26    D3 5F       OUT  5F        メインRAMにバンク切り換え。
BD28    C9          RET            リターン。
BD29    D5          PUSH D         DEの値をスタックへ退避。
BD2A    0E 10       MVI  C,10      C←10H……縦のライン数。
BD2C    06 04       MVI  B,04      B←4……横のバイト数。
BD2E    7E          MOV  A,M       A←(HL)……パターンデータ。
BD2F    12          STAX D         (DE)←A……G.V.RAMにパターンデータを
                                   入れる。
BD30    23          INX  H         HL←HL+1……次のデータアドレスにする。
BD31    13          INX  D         DE←DE+1……G.V.RAMを+1(右隣り)にする。
BD32    10 FA       DJNZ BD2E      B←B−1し,B≠0ならBD2EH番地へジャンプ。
```

BD34	E5	PUSH H	HLの値をスタックへ退避。
BD35	21 004C	LXI H,004C	DE←DE+4CHを実現。
BD38	19	DAD D	DEを次のラインの先頭アドレスにする。
BD39	EB	XCHG	
BD3A	E1	POP H	HLの値をスタックから取り出す。
BD3B	0D	DCR C	C←C−1.
BD3C	20 EE	JRNZ BD2C	C≠0ならBD2CH番地へジャンプ。
BD3E	D1	POP D	DEの値をスタックから取り出す。
BD3F	C9	RET	リターン。

LIST 4-4

D000	F3	DI	割込禁止。
D001	01 0000	LXI B,0000	BC← 0000 H.
D004	3E 00	MVI A,00	A←0.
D006	CD BD00	CALL BD00	(C,B)にパターン番号Aを表示。
D009	FB	EI	割込み許可。
D00A	FF	RST 7	h]の状態へ戻る。

■レジスタのやりくりに注目

このプログラムの中で、STAX Dというニモニックが新しく出ていますが、Xはペアレジスタのことですから、ここではDEレジスタになります。STAは、ご存じのようにSTore Accumulatorの略で、つまりDEレジスタで示されるアドレスの中にアキュムレータの値を入れる、という意味です。リスト中の説明にもありますが、このようにアドレスやレジスタで示されるものの中身を表す時には、(　)で囲んで表現します。

プログラム的には、これまでグラフィックV.RAMに入れていたFF_Hを、M(HLレジスタで示されたアドレスの中身、すなわちパターンデータ)に変えて、DEレジスタで示されるグラフィックV.RAMアドレスに入れるよう

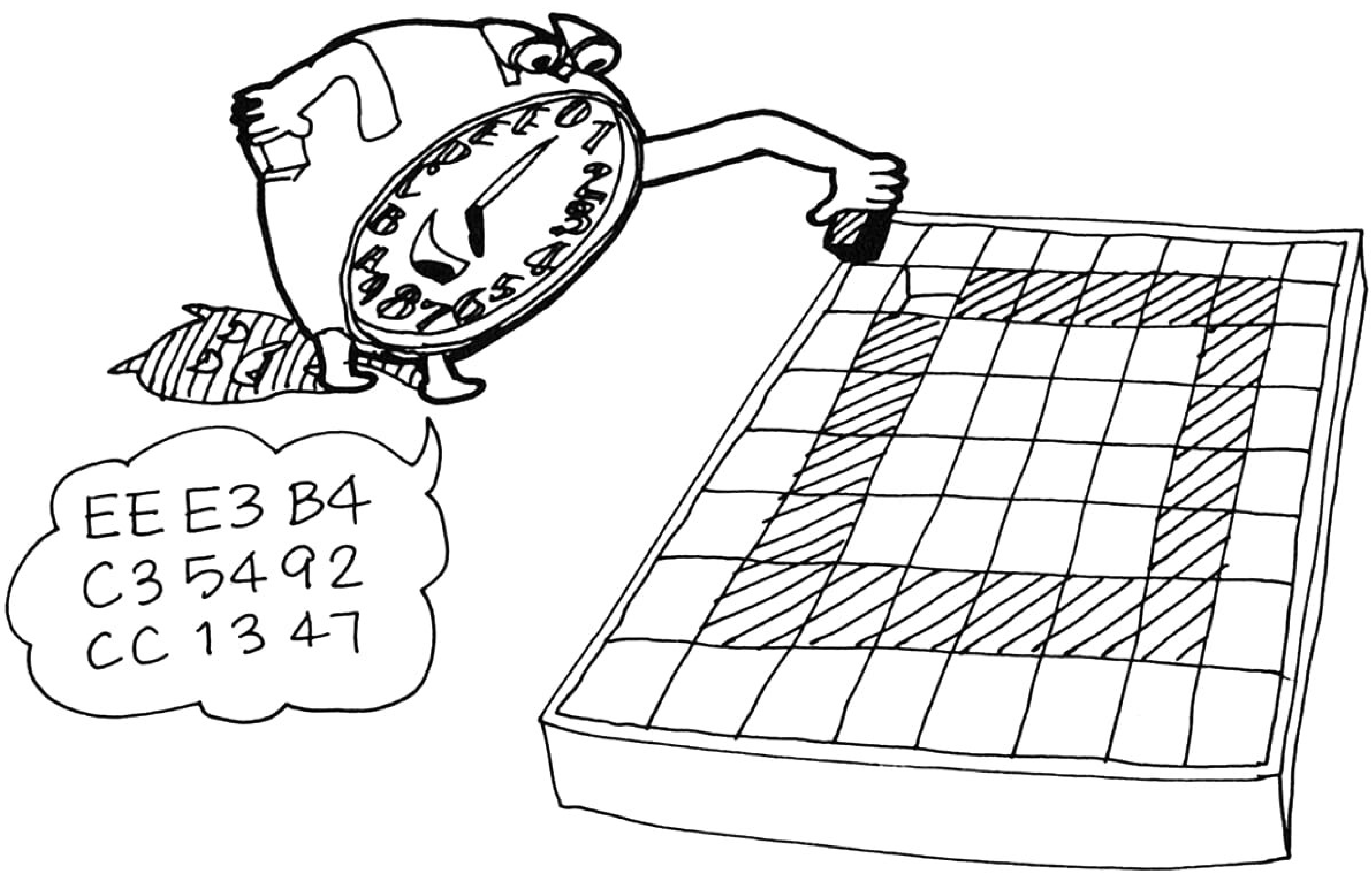
にしただけのことです。とくにアルゴリズムで難しい点はないので、簡単に理解できると思いますが、DE＝DE＋$4C_H$を実現する時のレジスタのヤリクリなどをおぼえるようにしてください。

■サブプログラムを実行してみる

パターンデータがなくても、00_H以外の数値がデータアドレスに入っていれば、何か表示されるはずですから、サブプログラムのテストを実行してみましょう。

```
h]GD000↵
h]▮
```

サブのBCレジスタやアキュムレータの値を変えての実験も、当然のことですがやってみてください。また、本物のパターンデータがなくても、Eコマンドで適当なデータを作って表示させるのも面白いかもしれません。



4-3 パターンデータの自動作成

画面にパターンを表示するプログラム――マシン語マスターの成果としては立派なものといえるかもしれませんが、これだけでは宝の持ちぐされとなってしまいます。やはり、実際のパターンデータで表示させてこそ、プログラムも生きてくるというものです。そのためには、パターンデータを作成するという地道な作業も避けて通るわけにはいきません。

ただ、データを作るといっても、パターンを色鉛筆で方眼紙に描いて、それをB・R・Gのデータに分解していく、というのではあまりにも大変です。せっかく作ったマシン語プログラムも、これでは利用する前にヤル気がなくなってしまいそうです。

■パターンエディタ

そこで登場するのが、**パターンエディタ**というデータ自動作成のツールです。つまり、これはパターン表示の逆のことをしてくれるプログラムで、画面上にあるパターンのデータをグラフィックV.RAMから読み取り、パターンデータのあるべきメモリに入れてくれるというものです。

では、そのパターンはどのようにして作成するかというと、BASICのPSET命令により、ドットを1つずつ打って作っていくしかありません。といっても、その程度のことはキー1つでできるようになっていますから安心です。

なお、このパターンエディタで使用されているマシン語部の内容は、指定のパターンのデータ化と、メモリにある6種のパターンを一度に表示することです。

マシン語プログラムについては、BASICプログラム上にDATA文として置いてあるので、直接入力する必要はありませんが、参考までに本章の最後

（第５節）にリストとしても載せてあります。

■パターンエディタの使い方

プログラムの入力を終えたら、走らせる前に必ずセーブしてください。また、DATA 文はマシン語プログラムですから、１つでも間違えると暴走したり、正しく動作しなくなる危険性があります。注意して入力しましょう。プログラムを走らせると、まず作成するパターンのサイズを聞いてきます。

```
パターン　サイズ　(Max＝32, 16)
＝▮
```

これには「横のドット数、縦のドット数」と入力しますが、最大は横32ドット、縦16ドットとなっています。横のドット数は、データが８ドット単位で作られますから、８の倍数で入力するようにしてください。ここでは、32, 16 ⏎ とします。画面がパターンエディタ・モードに変わります。

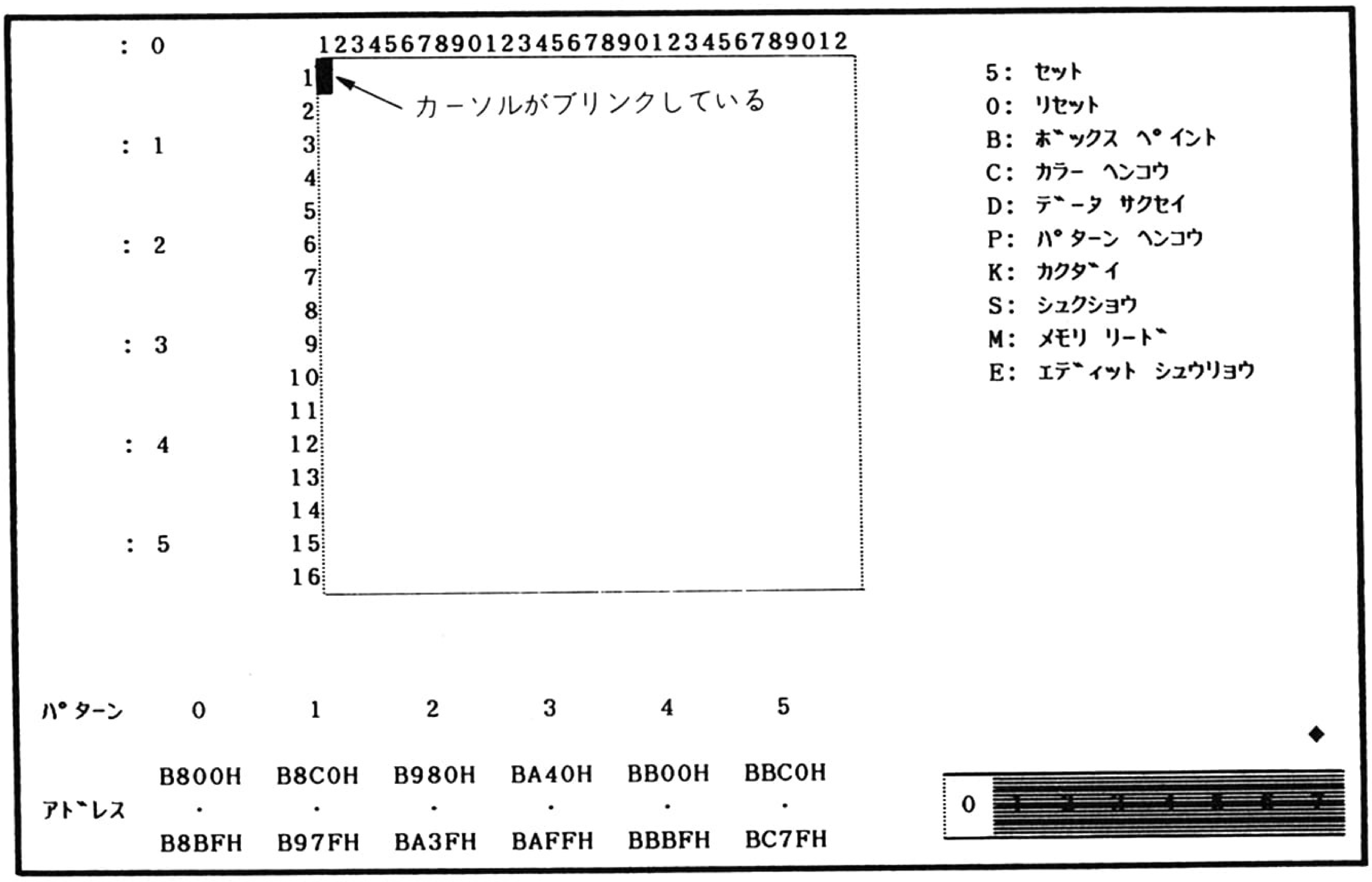


図4.3　パターンエディット・モード

■キーに割りあてた機能

パターンの作成は、0～5までの6パターンを、どれでも自由にエディットできるようになっています。画面左は作成したパターンが表示される部分で、現在エディット中のパターン番号は赤で示されています。また、各パターンのデータが入るアドレスが画面下部に表示されていますが、データになるのはDを押してからのことで、エディットしただけではデータにはなりません。その他の機能は、**図4.4**の通りです。

■実験

試しにMを押してみましょう。データエリアには、たぶん00_H以外の数値が入っているでしょうから、ピッという音とともに何らかのパターンが6つ表示されたはずです。

つぎにKを押すと、パターン番号0が拡大表示されます。この方法により、いったん作成してセーブしたパターンでも、データをメモリに再ロードすれば、何度でも修正ができることになります。同様に、テンキーを使ってパターンを修正したあとで、やはりもとにもどしたくなってしまったという時などにもMで修復することができるので、うまく利用してください。

また、KとSを使えば一部分だけ違うパターンを作ったり、パターンを別の番号の所に入れたりすることが簡単にできるようになります。

エディットし終えたパターンは、必ずDによりデータをメモリに落とさなければなりません。Dを押すと、ピッという音とともにそのパターン番号とデータアドレスの部分が赤く光り、メモリに入ったことを知らせてくれます。

■パターンをつくる

このパターンエディタは、あくまでもマシン語マスターのための補助ツールですから、機能的には大したことはありません。それでも、並みのゲームの1本や2本を作ることは十分に可能なだけの能力は秘めています。

テスト使用の意味も込めて、早速つぎのようなパターンを作成してください。作成するパターンは、口絵の3タイプの中から好みの1つを選んで、パターン番号の0と1でエディットしてください。

機　能	対応するキー
ドットをセットする	5
ドットをリセットする	0
カーソルを移動する	
上（↑）	8
下（↓）	2
左（←）	4　いずれもテンキー
右（→）	6
左上（↖）	7
左下（↙）	1
右上（↗）	9
右下（↘）	3
カーソル移動と同時にドットをセットとする	SHIFT と上記のカーソル移動キーを同時に押す
現在色で塗りつぶす	B
セットする色を変更する	C
エディット中のパターンデータを作成する	D
エディットするパターンを変更する	P　いずれも
画面左のパターンを中央に拡大する	K　JIS配列キー
画面中央の拡大パターンを画面左に縮小表示する	S
メモリ上のデータで画面左に6パターン表示する	M
プログラムを終了する	E

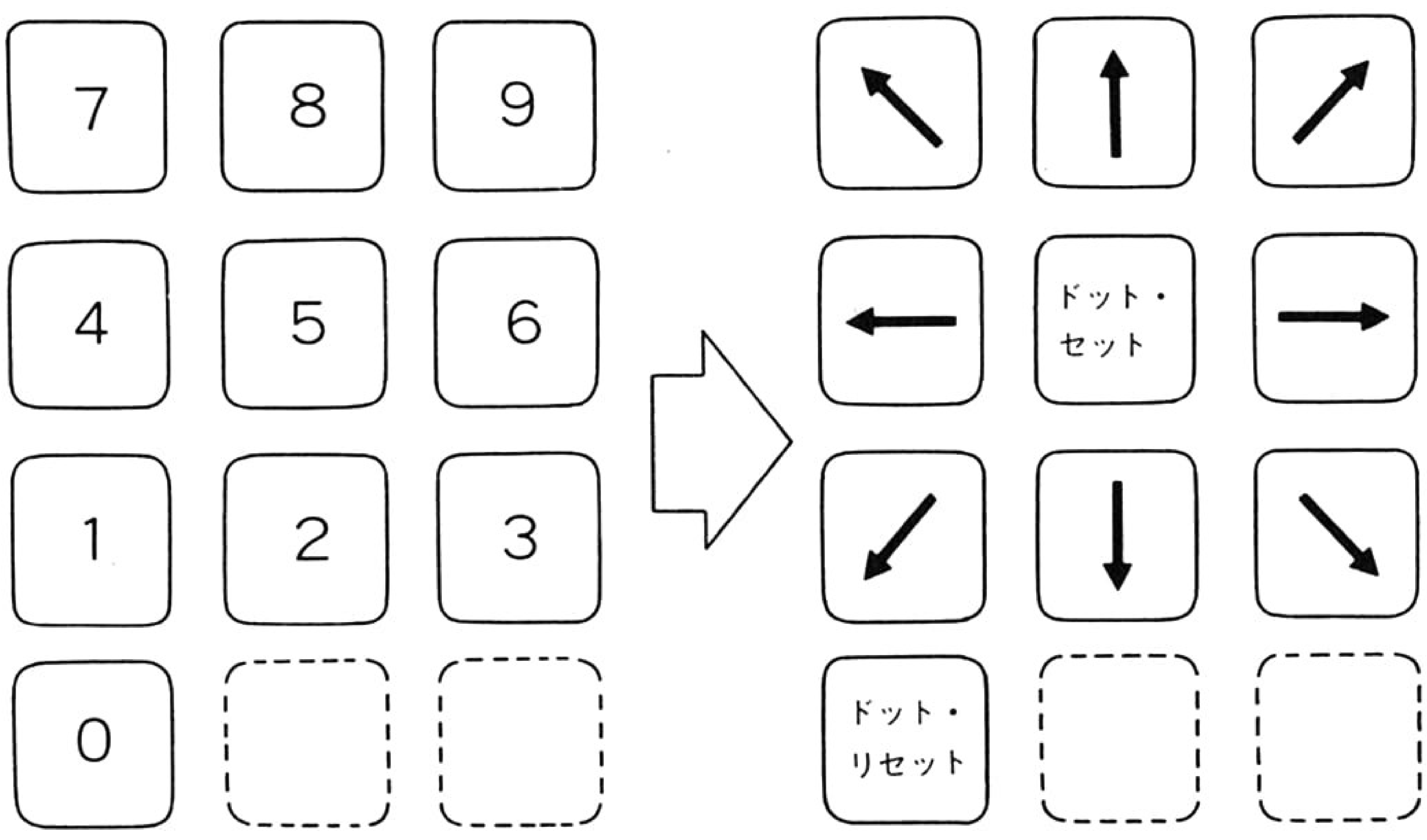
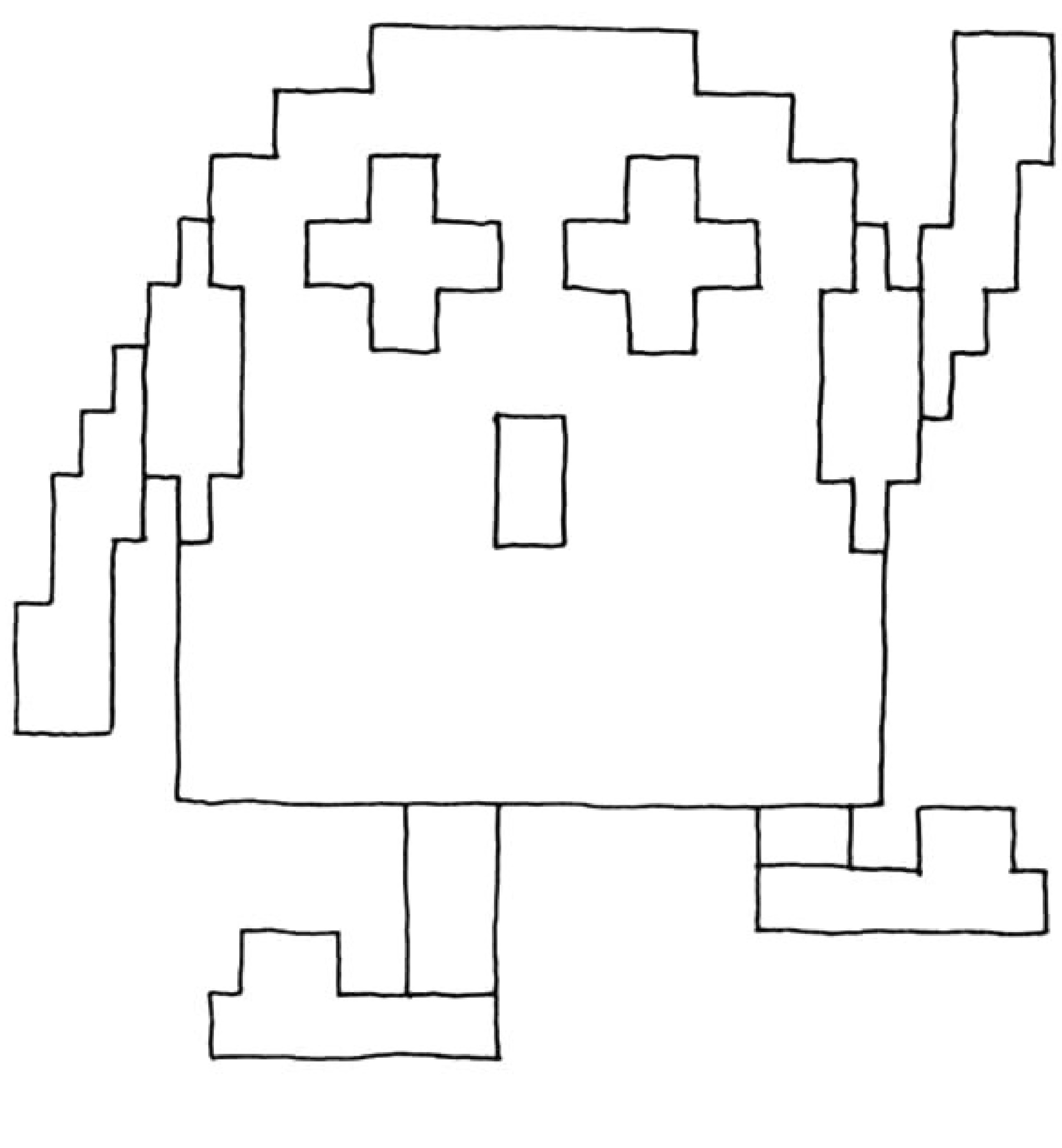


図4.4　パターンエディター操作表

エディットを終えたら、それぞれDを押してデータにし、Eによりプログラムを終了します。

最後に、パターンデータをディスクなりテープにセーブするのも忘れないように。



LIST 4-5

```
1000 'PATTERN EDITOR by T.H on 1986.1.20
1010 CLEAR,&HB7FF:DEFINT A-Z:SCREEN 0,3
1020 WIDTH 80,25:CONSOLE 0,25,0,1
1030 FOR N=0 TO 166
1040 READ A$:POKE &HBF00+N,VAL("&H"+A$)
1050 NEXT
1060 CLS:PRINT"ﾊﾟﾀｰﾝ ｻｲｽﾞ (Max=32,16)"
1070 PRINT:INPUT"= ",MX,MY
1080 IF MX<1 OR MX>32 THEN 1060
1090 IF MY<1 OR MY>16 THEN 1060
1100 POKE &HBF07,(MX-1)¥8+1
1110 POKE &HBF06,MY:CLS:SCREEN 0,0
1120 DEF USR=&H4290:DIM CL%(117)
1130 LINE(432,180)-(624,195),7,B
1140 FOR N=0 TO 6
1150 A=456+N*24:LINE(A,180)-(A,195),7
1160 NEXT
1170 FOR N=0 TO 7
1180 A=433+N*24:LINE(A,181)-(A+22,194),N,BF
1190 IF N>0 THEN COLOR 0
1200 LOCATE 54+N*3,23:PRINT N:A=433+N*24
1210 GET(A,181)-(A+6,187),CL%(13*N)
1220 NEXT
1230 COLOR 7:LOCATE 76,21:PRINT"♦"
1240 LOCATE 17,0
1250 FOR N=1 TO MX
1260 PRINT HEX$(N MOD 10);
1270 NEXT
1280 FOR N=1 TO MY
1290 LOCATE 15,N:PRINT USING"##";N;
1300 NEXT
```

```
1310 LINE(136,7)-(138+MX*8,MY*8+7),7,B
1320 LOCATE 57,1:PRINT"5: ｾｯﾄ"
1330 LOCATE 57,2:PRINT"Ø: ﾘｾｯﾄ"
1340 LOCATE 57,3:PRINT"B: ﾎﾞｯｸｽ ﾍﾟｲﾝﾄ"
1350 LOCATE 57,4:PRINT"C: ｶﾗｰ ﾍﾝｺｳ"
1360 LOCATE 57,5:PRINT"D: ﾃﾞｰﾀ ｻｸｾｲ"
1370 LOCATE 57,6:PRINT"P: ﾊﾟﾀｰﾝ ﾍﾝｺｳ"
1380 LOCATE 57,7:PRINT"K: ｶｸﾀﾞｲ"
1390 LOCATE 57,8:PRINT"S: ｼｭｸｼｮｳ"
1400 LOCATE 57,9:PRINT"M: ﾒﾓﾘ ﾘｰﾄﾞ"
1410 LOCATE 57,10:PRINT"E: ｴﾃﾞｨｯﾄ ｼｭｳﾘｮｳ"
1420 FOR N=Ø TO 5
1430 LOCATE 5,N*3:PRINT":";N
1440 NEXT:COLOR@(6,Ø)-(7,Ø),2
1450 LOCATE Ø,20:PRINT"ﾊﾟﾀｰﾝ"
1460 LOCATE Ø,23:PRINT"ｱﾄﾞﾚｽ"
1470 FOR N=Ø TO 5
1480 LOCATE 8+N*7,20:PRINT N
1490 LOCATE 7+N*7,22
1500 PRINT HEX$(&HB8ØØ+N*&HCØ);"H"
1510 LOCATE 9+N*7,23:PRINT"･"
1520 LOCATE 7+N*7,24
1530 PRINT HEX$(&HB7FF+N*&HCØ+(MX¥8)*MY*3);"H";
1540 NEXT
1550 CX=17:CY=1:CC=7:PT=Ø:POKE &HBDØ8,PT
1560 *LOOP
1570 LOCATE CX,CY:A$=INPUT$(1)
1580 ON INSTR("246813795Ø",A$) GOSUB
     *D2,*D4,*D6,*D8,*D1,*D3,*D7,*D9,*ST,*RS
```

```
1590 IF A$="B" OR A$="b" THEN *B.PAINT
1600 IF A$="C" OR A$="c" THEN *C.CHANGE
1610 IF A$="D" OR A$="d" THEN *D.MAKE
1620 IF A$="P" OR A$="p" THEN *P.CHANGE
1630 IF A$="K" OR A$="k" THEN *LARGE
1640 IF A$="S" OR A$="s" THEN *SMALL
1650 IF A$="M" OR A$="m" THEN *MEMORY
1660 IF A$="E" OR A$="e" THEN *P.END
1670 IF (INP(8) OR 191)=191 THEN GOSUB *ST
1680 GOTO *LOOP
1690 *D2
1700 IF CY<MY THEN CY=CY+1
1710 RETURN
1720 *D4
1730 IF CX>17 THEN CX=CX-1
1740 RETURN
1750 *D6
1760 IF CX<MX+16 THEN CX=CX+1
1770 RETURN
1780 *D8
1790 IF CY>1 THEN CY=CY-1
1800 RETURN
1810 *D1
1820 IF CX=17 OR CY=MY THEN RETURN
1830 CX=CX-1:CY=CY+1:RETURN
1840 *D3
1850 IF CX=MX+16 OR CY=MY THEN RETURN
1860 CX=CX+1:CY=CY+1:RETURN
1870 *D7
1880 IF CX=17 OR CY=1 THEN RETURN
1890 CX=CX-1:CY=CY-1:RETURN
```

```
1900 *D9
1910 IF CX=MX+16 OR CY=1 THEN RETURN
1920 CX=CX+1:CY=CY-1:RETURN
1930 *ST
1940 PUT@(CX*8+2,CY*8),CL%(CC*13),PSET
1950 PSET(CX-17,PT*24+CY-1),CC:RETURN
1960 *RS
1970 PUT@(CX*8+2,CY*8),CL%(0),PSET
1980 PSET(CX-17,PT*24+CY-1),0:RETURN
1990 *B.PAINT
2000 FOR N=1 TO MY
2010 FOR M=1 TO MX
2020 PSET(M-1,24*PT+N-1),CC
2030 A=USR(0):LOCATE 16+M,N
2040 PUT@(130+M*8,N*8),CL%(CC*13),PSET
2050 NEXT:NEXT:GOTO *LOOP
2060 *C.CHANGE
2070 LOCATE 64,4:PRINT"No. 0-7 ? ";
2080 A=USR(0):A$=INKEY$
2090 IF A$<"0" OR A$>"7" THEN 2080
2100 LOCATE 64,4:PRINT"ﾍﾝｺｳ        "
2110 LOCATE 55+CC*3,21:PRINT" "
2120 CC=VAL(A$)
2130 LOCATE 55+CC*3,21:PRINT"♦"
2140 GOTO *LOOP
2150 *D.MAKE
2160 BEEP 1
2170 COLOR@(7+PT*7,20)-(11+PT*7,24),2
2180 DEF USR1=&HBF00:A=USR1(0)
```

```
2190 FOR N=0 TO 100:NEXT
2200 COLOR@(7+PT*7,20)-(11+PT*7,24),7
2210 BEEP 0:GOTO *LOOP
2220 *P.CHANGE
2230 LOCATE 66,6:PRINT"No. 0-5 ? ";
2240 A=USR(0):A$=INKEY$
2250 IF A$<"0" OR A$>"5" THEN 2240
2260 LOCATE 66,6:PRINT"ヘンコウ       "
2270 COLOR@(7,PT*3)-(7,PT*3),7
2280 PT=VAL(A$):POKE &HBF08,PT
2290 COLOR@(7,PT*3)-(7,PT*3),2
2300 GOTO *LOOP
2310 *LARGE
2320 FOR N=1 TO MY
2330 FOR M=1 TO MX
2340 C1=POINT(M-1,24*PT+N-1)
2350 A=USR(0):LOCATE 16+M,N
2360 PUT@(130+M*8,N*8),CL%(C1*13),PSET
2370 NEXT:NEXT:GOTO *LOOP
2380 *SMALL
2390 FOR N=1 TO MY
2400 FOR M=1 TO MX
2410 C1=POINT(130+M*8,N*8)
2420 A=USR(0):LOCATE 16+M,N
2430 PSET(M-1,24*PT+N-1),C1
2440 NEXT:NEXT:GOTO *LOOP
2450 *MEMORY
2460 BEEP 1:DEF USR1=&HBF03:A=USR1(0)
2470 BEEP 0:GOTO *LOOP
2480 *P.END
```

```
2490 LOCATE 0,24:END
2500 DATA C3,09,BF,C3,53,BF,10,04,00,F3
2510 DATA 21,40,B7,11,C0,00,3A,08,BF,3C
2520 DATA 47,19,10,FD,E5,47,21,80,B8,11
2530 DATA 80,07,19,10,FD,D1,D3,5C,CD,37
2540 DATA BF,D3,5D,CD,37,BF,D3,5E,CD,37
2550 DATA BF,D3,5F,FB,C9,E5,3A,06,BF,4F
2560 DATA 3A,07,BF,47,E5,7E,12,23,13,10
2570 DATA FA,E1,D5,11,50,00,19,D1,0D,20
2580 DATA EB,E1,C9,F3,AF,F5,CD,61,BF,F1
2590 DATA 3C,FE,06,20,F6,FB,C9,3C,47,21
2600 DATA 80,B8,11,80,07,19,10,FD,E5,47
2610 DATA 21,40,B7,11,C0,00,19,10,FD,D1
2620 DATA D3,5C,CD,8A,BF,D3,5D,CD,8A,BF
2630 DATA D3,5E,CD,8A,BF,D3,5F,C9,D5,3A
2640 DATA 06,BF,4F,3A,07,BF,47,D5,7E,12
2650 DATA 23,13,10,FA,D1,E5,21,50,00,19
2660 DATA EB,E1,0D,20,EA,D1,C9
```

4-4 パタパタ・アニメ

前節で作成したパターンを見て、学校の教科書やノートの端に漫画を描いてパタパタさせたことを思い出した人もいるかもしれません。あれは、まさにアニメーションの基本であり、今日のこの日のための予行演習であったといえます。画面の中をチョコチョコ動いてくれるかわいいキャラクターも、分解すればすべていくつかのパターンになってしまうわけです。ここでは、作成したばかりのパターンを実際にパタパタ・アニメにして、画面上で見ることにしましょう。

すでに「(C, B) にパターンAを表示する」というルーチンがあるわけですから、プログラムの考え方としてはAの値を0と1交互にして表示すればいいことがわかります。ということは、**LIST4-4**にパターン1を表示するプログラムを追加して、何回かループさせればいいということです。

結果だけを追うのであればそれでもいいのですが、これではマシン語のマスターのためにわざわざパターンエディタを使って、立派なパターンを作った意味がなくなってしまいます。そこで、このような簡単な例を機会に、ワークエリアの使い方もおぼえてしまおう、というのがこの節での目的です。

■ワークエリアの使い方

まず、ここで用いるワークエリアのアドレスと内容ですが、$C500_H$番地からつぎのように設定してあります。

$C500_H$番地：パターンのX座標
$C501_H$番地：パターンのY座標
$C502_H$番地：パターン番号

実際には、座標は常に（0，0）に固定なのですが、将来への布石のため、表示の都度ワークエリアから読んでいきます。

■ウエイトを取る

ところで32×16程度のサイズでは、パターンはそれこそ一瞬にして表示されてしまいますから、単に2パターンを交互に表示させたのでは、とても人間の目には動いているようには見えません。パターンが動いて見えるためには、人間の目に残像として残るだけの時間が必要です。これには無駄ループといって、その場で駆け足足踏みするような時間かせぎ命令を与えます。

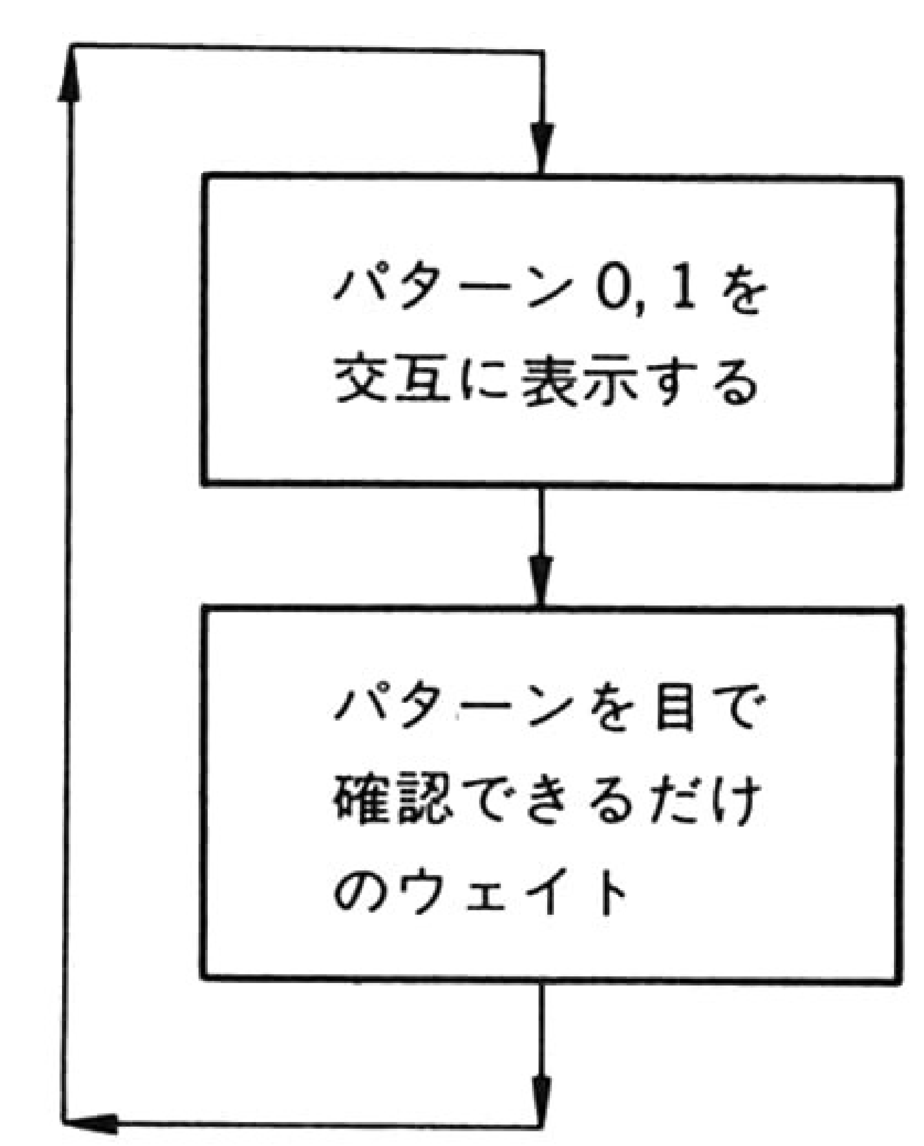


図4.5　パソコンでのパタパタ・アニメ

たかだか、2パターンを使ったパタパタ・アニメでも、ここまで気を使わなければならないのですから、マシン語とは面倒なものですね。しかし、この**ウエイト**を取るという感覚は、ゲームにおいては非常に重要なポイントとなっているからやっかいなのです。

本書で扱っているような入門初心者段階では関係のないことですが、たとえばインベーダーのようなゲームで、画面に敵がたくさん出現している時と残り少ない時を考えると、当然のことながら画面表示に要する時間が違ってくるわけです。そんな場合に、主人公や弾の動くスピードを一定に保つ役目をするのがウエイトなのです。タイマーという場合もあります。

というようなことを暗に含みながら、**LIST4-6**を見るようにすると、いわゆる無駄命令も無駄にはならなくなってきます。

プログラムは、最初にワークエリアへ初期値を入れることから始まっています。そして、パターンを表示してからパターン番号の変更（1なら0へ、0なら1へ）を行い、ついでCレジスタとBレジスタを使って、無駄ループによるウエイトを取っています。これを、トータルで50_H回（十進数で80回）繰り返しているわけです。

■CPI

ニモニックを見ると、パターンを表示する前後でPUSH　PSW/POP PSW、つまりアキュムレータとフラグのPUSH/POPを行っていますが、これはアキュムレータで示されるパターン番号を保持したいためで、フラグはここでは関係ありません。そして、もう1つの新しいニモニックが、そのあとに出てきています。

ニモニック （読み方）	説　　　明
CPI 01 （コンペア イミーディエート）	CPIはComPare Immediateの略で、アキュムレータの値とCPIに続く数値とを比較（アキュムレータの値－数値）し、その結果をフラグに反映させる命令。数値以外にも、CMP＋レジスタ（Mを含む）として同様の比較判定ができる。フラグ以外の値は変化しない。

BASICでもマシン語でも、本格的なまともなプログラムであれば、そこには必ず条件分岐が含まれています。そして、マシン語の場合の条件はすべてフラグによる判定であるということは、すでに説明した通りです。

これまでのプログラムではフラグの変化を条件分岐前の減算によって行ってきましたが、当然のことながら、これはレジスタの値が変化していい場合だけに限られていました。そこで、アキュムレータの値から仮想的に減算をしてフラグだけを変化させる、という機能を持たせたのが、このCompare（比

LIST 4-6

```
D000   F3         DI          割込み禁止。
D001   3E 00      MVI  A,00   A←0.
D003   32 C500    STA  C500   (C500H)←A……表示X座標。
D006   32 C501    STA  C501   (C501H)←A ……表示Y座標。
D009   32 C502    STA  C502   (C502H)←A ……〃パターン番号。
D00C   06 50      MVI  B,50   B←50H……パタパタアニメの回数。
D00E   C5         PUSH B      BCの値をスタックへ退避。
D00F   3A C500    LDA  C500   A←(C500H)……表示X座標。
D012   4F         MOV  C,A    C←A.
D013   3A C501    LDA  C501   A←(C501H)……表示Y座標。
D016   47         MOV  B,A    B←A.
D017   3A C502    LDA  C502   A←(C502H)……パターン番号。
D01A   F5         PUSH PSW    PSW(アキュムレータ&フラグ)の値をスタッ
                              クへ退避。
D01B   CD BD00    CALL BD00   BD00H番地をコール(C,B)にAを表示。
D01E   F1         POP  PSW    PSWの値をスタックから取り出す。
D01F   FE 01      CPI  01     }
D021   28 03      JRZ  D026   } A＝1ならD026H番地へジャンプ。
D023   3C         INR  A      A←A＋1……A＝1となる。
D024   18 01      JMPR D027   D027H番地へジャンプ。
D026   3D         DCR  A      A←A－1……A＝0となる。
D027   32 C502    STA  C502   (C502H)← A……パターン番号をメモリに
                              格納。
D02A   0E 90      MVI  C,90   }
D02C   06 FF      MVI  B,FF   } ウェイト……無駄命令によるウェイト。
D02E   10 FE      DJNZ D02E   }
D030   0D         DCR  C      } 90H回『B←B－1をFFH回』を実行。
D031   20 F9      JRNZ D02C   }
D033   C1         POP  B      BCの値をスタックから取り出す。
D034   10 D8      DJNZ D00E   パタパタアニメをB回繰り返す。
D036   FB         EI          割込み許可。
D037   FF         RST  7      h]の状態へ戻る。
```

較）命令なのです。

まだ、具体的な例としては出ていませんが、アキュムレータからの減算にはSubtract命令があり、レジスタの値や任意の数値をマイナスすることができます。

つまり、Compare命令とはレジスタの値が変化しない減算命令のことで、フラグだけが必要な条件分岐には欠かせない存在というわけです。

■アニメの実行

それでは、実際にプログラムを走らせてみましょう。実行に際しては、パターン表示ルーチンとパターンデータをロードするのを忘れてはなりません。パタパタするスピード（つまりウエイトを取る時間）は機種やディスプレイ装置によっても違ってきますので、ウエイト時間を調節する値であるCレジスタの数値を変えて楽しんでください。



4-5 論理演算の利用法

パタパタ・アニメで楽しんだあとには、ふたたびマシン語らしい世界に入ります。これからやるのは、**論理演算**という、数学のきらいな人には身の毛もよだつようなことです。ところが、何でもそうなのですが、学問としてやると難しいことでも、現実に利用価値があるとなると意外にわかるということがあるものです。第一、ここは高度な数学の世界などではなく、かけ算・わり算もできないマシン語の世界なのです。どうせ低レベルのことに違いないとみてよさそうです。

■パターンエディタのマシン語リスト

さて、まずは第3節で作成したパターンエディタのマシン語部のリストを、約束通り載せることにします。少し長いリストですが、これに関しては実行される内容も、またニモニックについても、99パーセントはすでに説明ずみでわかっているはずですから、各自で理解してもらうことにします。

この中で、$BF06_H$～$BF08_H$はニモニックとしての表示もありますが、これはワークエリアの初期値をアセンブラが勝手にニモニックにしているだけのことです。命令としては何の意味もないことですから、注意してください。

また、PUSH/POPなどで、これまでにはなかった使い方をしている所もありますので、レジスタの用法としておぼえておくと便利です。

ここで読者が疑問に思うことは、説明もしないプログラムをなぜわざわざここに置くのか、ということではないでしょうか。

その答えは、じつはこのリストの中に、本書ではじめての論理演算命令が出ているからなのです。$BF54_H$番地にあるXRA A――これが問題の論理演算ですが、ここでの使用目的はアキュムレータの値を0にするというだけのことです。

LIST 4-7-1

```
BF00   C3 BF09    JMP   BF09      BF09H番地(データ作成ルーチン)へジャンプ。
BF03   C3 BF53    JMP   BF53      BF53H番地(6種のパターンを表示)へジャンプ
BF06   10 04      DJNZ  BF0C      ワーク (BF06H)=縦のライン数,
                                        (BF07H)=横のバイト数。
BF08   00         NOP             エリア (BF08H)=パターン番号。
BF09   F3         DI              割込み禁止。
BF0A   21 B740    LXI   H,B740    HL←B740H.
BF0D   11 00C0    LXI   D,00C0    DE←00C0H.
BF10   3A BF08    LDA   BF08      A←(BF08H)……パターン番号。
BF13   3C         INR   A
BF14   47         MOV   B,A       HLにデータアドレスを求める。
BF15   19         DAD   D
BF16   10 FD      DJNZ  BF15
BF18   E5         PUSH  H         HLの値をスタックへ退避。
BF19   47         MOV   B,A
BF1A   21 B880    LXI   H,B880    HLにパターンの表示されているアドレス
BF1D   11 0780    LXI   D,0780    を求める。
BF20   19         DAD   D
BF21   10 FD      DJNZ  BF20
BF23   D1         POP   D         スタックの値をDEレジスタに取り出す。
BF24   D3 5C      OUT   5C        ブルー面にバンク切り換え。
BF26   CD BF37    CALL  BF37      BF37H番地をコール(データ作成)。
BF29   D3 5D      OUT   5D        レッド面にバンク切り換え。
BF2B   CD BF37    CALL  BF37      BF37H番地をコール(データ作成)。
BF2E   D3 5E      OUT   5E        グリーン面にバンク切り換え。
BF30   CD BF37    CALL  BF37      BF37H番地をコール(データ作成)。
BF33   D3 5F      OUT   5F        メインRAMにバンク切り換え。
BF35   FB         EI              割込み許可。
BF36   C9         RET             BASICへ戻る。
BF37   E5         PUSH  H         HLの値(パターンの表示されているアドレス)を
                                    スタックへ退避。
BF38   3A BF06    LDA   BF06      A←(BF06H)………縦のライン数。
BF3B   4F         MOV   C,A       C←A.
BF3C   3A BF07    LDA   BF07      A←(BF07H)………横のバイト数。
```

BF3F	47	MOV　B,A	B←A.
BF40	E5	PUSH　H	HLの値をスタックへ退避。
BF41	7E	MOV　A,M	A←(HL) }G.V.RAMの値をDEで示されるデータアドレスに入れる。
BF42	12	STAX　D	(DE)←A
BF43	23	INX　H	HL←HL＋1……G.V.RAMを＋1(右隣り)にする。
BF44	13	INX　D	DE←DE←DE＋1……データアドレスを＋1する。
BF45	10 FA	DJNZ　BF41	横のバイト数だけBF41Hから以降を繰り返す。
BF47	E1	POP　H	HLの値をスタックから取り出す。
BF48	D5	PUSH　D	DEの値をスタックへ退避。
BF49	11 0050	LXI　D,0050	}HL←HL＋50Hを実現(次ラインのアドレス)。
BF4C	19	DAD　D	
BF4D	D1	POP　D	DEの値をスタックから取り出す。
BF4E	0D	DCR　C	}縦のライン数だけBF3CH以降を繰り返す。
BF4F	20 EB	JRNZ　BF3C	
BF51	E1	POP　H	HLの値をスタックから取り出す。
BF52	C9	RET	リターン。

■MVI A, 00としないわけ

アキュムレータの値がなぜ0になるのか、それを考える前に、なぜ単純にMVI A, 00としなかったのか、その理由をちょっと知ってください。

理由：MVI A, 00ではメモリを2バイト使うが、XRA Aなら1バイトですむ。したがってメモリ効率がいいし、その分実行スピードもアップする。

なんというか、ケチのかたまりのように感じるかもしれませんが、これはいわば慣用句として論理演算が使われる例なのです。論理演算においては、実際に論理演算が目的で実行される場合と、このように単なる慣用句として一般的に用いられている場合とがあるということです。ただし、論理演算ではフラグの変化がありますから、慣用句といっても本来の用法と100パーセント同じになるというわけではありません。

LIST 4-7-2

BF53	F3	DI		割込み禁止。
BF54	AF	XRA	A	A←0……パターン番号。
BF55	F5	PUSH	PSW	PSW(アキュムレータ&フラグ)の値をスタックへ退避。
BF56	CD BF61	CALL	BF61	BF61H番地をコール(パターン番号Aを指定位置へ表示)。
BF59	F1	POP	PSW	PSWの値をスタックから取り出す。
BF5A	3C	INR	A	A←A+1.
BF5B	FE 06	CPI	06	A≠6なら$BF55_H$番地へジャンプ。
BF5D	20 F6	JRNZ	BF55	
BF5F	FB	EI		割込み許可。
BF60	C9	RET		BASICへ戻る
BF61	3C	INR	A	A←A+1.
BF62	47	MOV	B,A	B←A.
BF63	21 B880	LXI	H,B880	HLにパターン表示アドレスを求める。
BF66	11 0780	LXI	D,0780	
BF69	19	DAD	D	
BF6A	10 FD	DJNZ	BF69	
BF6C	E5	PUSH	H	HLの値をスタックへ退避。
BF6D	47	MOV	B,A	B←A.
BF6E	21 B740	LXI	H,B740	HLにパターンデータアドレスを求める。
BF71	11 00C0	LXI	D,00C0	
BF74	19	DAD	D	
BF75	10 FD	DJNZ	BF74	
BF77	D1	POP	D	スタックの値をDEレジスタに取り出す。
BF78	D3 5C	OUT	5C	ブルー面にバンク切り換え。
BF7A	CD BF8A	CALL	BF8A	$BF8A_H$番地をコール(四角形をデータで描く)。
BF7D	D3 5D	OUT	5D	レッド面にバンク切り換え。
BF7F	CD BF8A	CALL	BF8A	$BF8A_H$番地をコール(四角形をデータで描く)。
BF82	D3 5E	OUT	5E	グリーン面にバンク切り換え。
BF84	CD BF8A	CALL	BF8A	$BF8A_H$番地をコール(四角形をデータで描く)。
BF87	D3 5F	OUT	5F	メインRAMにバンク切り換え。
BF89	C9	RET		リターン。

BF8A	D5	PUSH D	DEの値をスタックへ退避。
BF8B	3A BF06	LDA BF06	A←(BF06H)……縦のライン数。
BF8E	4F	MOV C,A	C←A.
BF8F	3A BF07	LDA BF07	A←(BF07H)……横のバイト数。
BF92	47	MOV B,A	B←A.
BF93	D5	PUSH D	DEの値をスタックへ退避。
BF94	7E	MOV A,M	A←(HL)……パターンデータ。
BF95	12	STAX D	(DE)←A……G.V.RAMにパターンデータを入れる。
BF96	23	INX H	HL←HL+1……次のデータアドレスにする。
BF97	13	INX D	DE←DE+1……G.V.RAMを+1(右隣り)にする
BF98	10 FA	DJNZ BF94	横のバイト数だけBF94H以降を繰り返す。
BF9A	D1	POP D	DEの値をスタックから取り出す。
BF9B	E5	PUSH H	HLの値をスタックへ退避。
BF9C	21 0050	LXI H,0050	DE←DE+50Hを実現（次ラインのアドレス）。
BF9F	19	DAD D	
BFA0	EB	XCHG	
BFA1	E1	POP H	HLの値をスタックから取り出す。
BFA2	0D	DCR C	縦のライン数だけBF8FH以降を繰り返す。
BFA3	20 EA	JRNZ BF8F	
BFA5	D1	POP D	DEの値をスタックから取り出す。
BFA6	C9	RET	リターン。

■論理演算とは

では、論理演算とはどのようなものなのか、調べてみることにしましょう。

まず論理演算の種類ですが、これにはOR、AND、XORという3つの演算パターンがあります。どの演算命令も「アキュムレータの値」と「レジスタ値、または1バイトの数値」を二進数に分解し、ビットごとに演算を行うというものです。演算の結果はアキュムレータに入り、その結果にともなってフラグも変化します。それ以外のレジスタの値は、演算によって変化することはありません。

なお、演算というのは加減乗除を含め、数値を一定の規則にもとづいて操作することをいいますが、ほかの演算と同じように、結果よりも、その目的を知ることが理解への早道です。

(1) OR（論理和）

［ニモニック］**ORA＋レジスタ**　または　**ORI＋数値**

［意　味］演算をする2つの数値のうち、どちらかのビットでも1になっていればそのビットを1にし、両方とも0ならそのビットを0にするという命令。オアと読む。

［演算例］アキュムレータ＝$F0_H$、Cレジスタ＝55_HのORを取る。

$$
\begin{array}{rl}
F0_H = & 1\ 1\ 1\ 1\ \ 0\ 0\ 0\ 0 \\
\underline{\text{OR}\quad 55_H =} & \underline{0\ 1\ 0\ 1\ \ 0\ 1\ 0\ 1} \\
 & 1\ 1\ 1\ 1\ \ 0\ 1\ 0\ 1 = F5_H
\end{array}
$$

↑

アキュムレータの値となる

これは、おもにアキュムレータのある特定のビットを1にしたい時に用いられます。たとえば、アキュムレータの値が常に奇数になるようにしたければ、1とのORを取ればビット0だけが1となり、その他のビットは変化しないので、演算前のアキュムレータの値に近い奇数が得られるわけです。

また、慣用的な使われ方として、CPI 00（アキュムレータの値が0か否か）の代わりにORA Aはよく用いられます。つまり、アキュムレータ同士でORを取っても、全ビットがソックリ同じですから、できる数値には変化がなく、フラグだけが変化することになります。その結果、アキュムレータの値が0（もともとのアキュムレータの値が0の場合）ならば、ゼロフラグが立つというわけです。

(2) AND（論理積）

［ニモニック］**ANA＋レジスタ**　または　**ANI＋数値**

［意　味］演算をする2つの数値から、両方のビットが1になっているビットだけを1にし、それ以外のビットは0にするという命令。アンドと読む。

［演算例］アキュムレータ＝$F0_H$、1バイトの数値＝55_HのANDを取る。

$$
\begin{array}{r}
F0_H = 1\ 1\ 1\ 1\ \ 0\ 0\ 0\ 0 \\
\underline{AND\ 55_H = 0\ 1\ 0\ 1\ \ 0\ 1\ 0\ 1} \\
0\ 1\ 0\ 1\ \ 0\ 0\ 0\ 0 = 50_H
\end{array}
$$

↑
アキュムレータの値となる

これは、ORとは逆に、アキュムレータのある特定のビットを0にしたい時によく用いられます。たとえば、**LIST4-6**の$D01F_H$番地からの内容で、アキュムレータの値を1と0交互にしている部分がありますが、これは、じつはANDを使うとたった2つの命令ですんでしまうのです。

```
INR  A
ANI  1
```

あまりに簡単になりすぎて信じられないかもしれませんが、偶数・奇数というのは交互にならんでおり、これを二進数的にみると、ビット0の中身が0の時は偶数、1の時は奇数ということになります。1は二進数で0000 0001ですから、1とANDを取るとビット0以外はすべて0になってしまいます。つまり、1、2、3……という自然数列も、1とANDを取れば、それだけで1、0、1、0……となるということです。

この例では、アキュムレータの初期値を0とすると、つぎは1、そのつぎは +1されても1とANDを取られて0になり、結局、0、1、0、1……を繰り返すわけです。

同じような考え方からANI　7とすると、どんな数値でも0～7にすることができ、これはたとえばカラー番号を選ぶ時などに利用すると便利です。

(3) XOR（排他的論理和）

[ニモニック] **XRA＋レジスタ** または **XRI＋数値**

[意 味]演算する2つの数値のうち、一方のビットが1で、もう一方のビットが0になっているビットだけを1にし、それ以外のビットは0にするという命令。エクスクルーシブ オアと読む。

[演算例] アキュムレータ＝$F0_H$、Hレジスタ＝55_HのXORを取る。

```
       F0H＝1 1 1 1  0 0 0 0
   XOR 55H＝0 1 0 1  0 1 0 1
   ─────────────────────────
            1 0 1 0  0 1 0 1＝A5H
                               ↑
                     アキュムレータの値となる
```

これは、アキュムレータのある特定のビットだけを反転（1なら0に、0なら1に）させたいという目的でおもに使用されます。この命令の特徴は、二度繰り返すともとの値にもどるということで、この例でも$A5_H$をもう一度55_HとXORを取れば$F0_H$になりますので確認してみてください。

XOR命令はXRA A以外、マシン語に相当なれるまで用いる機会はないと思います。XRA Aは最初に書いた通り、アキュムレータの値を0にする時に慣用的に使用されますが、これは全ビット同じ数値同士でXORを取れば1になるビットがない(すなわち0になる)からです。ただし、MVI命令と違ってフラグが変化しますから、条件分岐がからむ時は注意が必要です。

■その他の論理演算命令

論理演算には、このほかにアキュムレータの全ビットを反転させるCMA (CoMplement Accumulatorの略）という命令もありますが、当面は使用することはないでしょう。

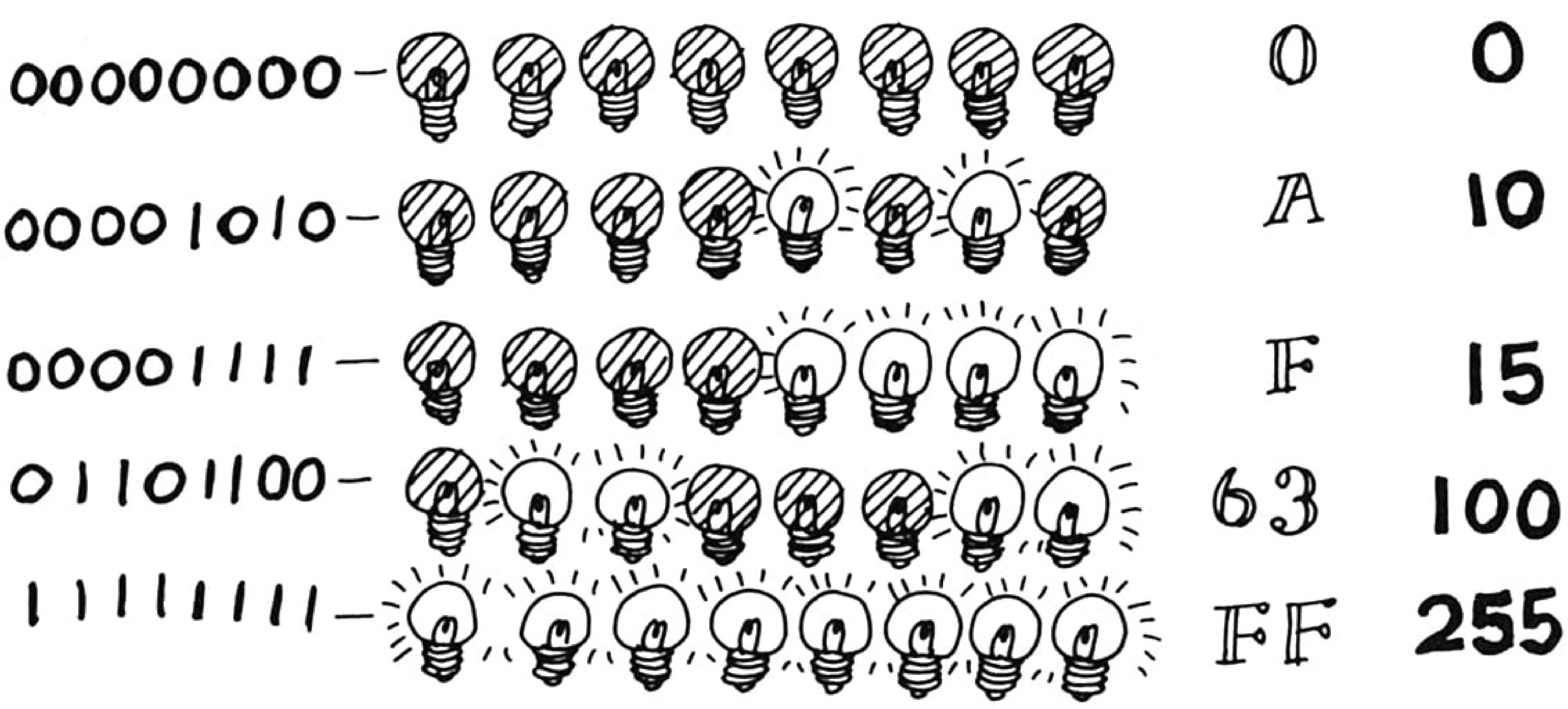
また、もう少しマシン語になれてくると、キャリーフラグの存在が大きくなってくる（キャリーフラグの値を含めた演算命令があるため）のですが、キャリーフラグのリセットの際にもORA AやAND Aが使われることがあるので、一応知っておいてください。

以上が、論理演算の概略です。ここですべてを理解しようとすると、なかなかつぎへ進まなくなってしまいますから、実際に論理演算に出会った時に、もう一度読み直すくらいの気楽さでいいかもしれません。

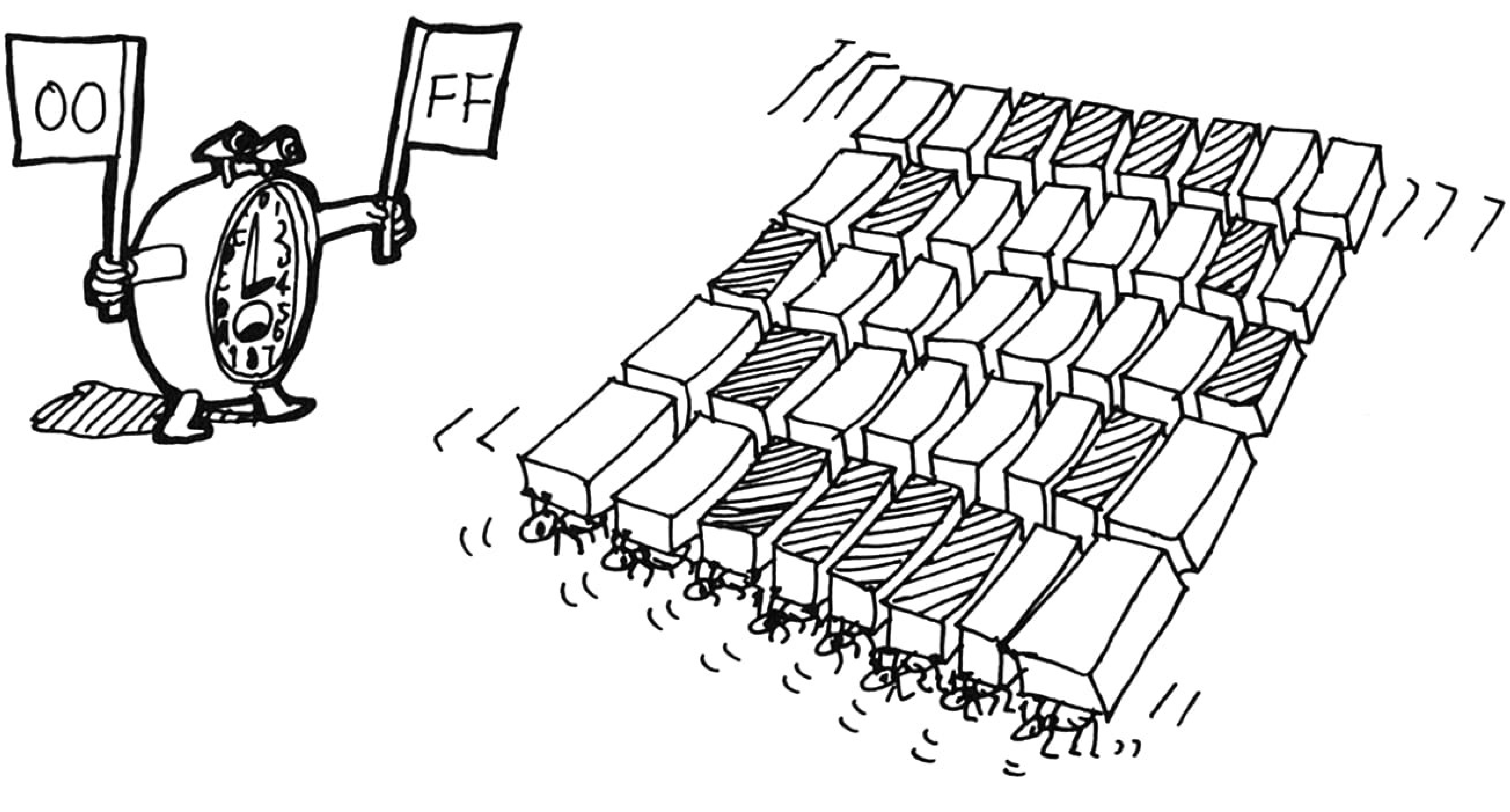
ただし、その時は単に実行結果だけを調べるのではなく、何のために論理演算が行われているのか、その目的をハッキリと理解することだけは忘れないでください。論理演算の目的がわかるようになったころには、マシン語が楽しくて仕方がない、という春のような気分になることでしょう。

《豆知識》８ビットの二進数・２桁の十六進数・十進数

８ビットの二進数の上位４ビットと下位４ビットを見ると、２桁の十六進数の上位と下位にきちんとあてはまっているのがわかる。



第5章
キャラクターパターン・コントロール



　これまでの進行状況から判断すると、本書はゲーム作りをしたい人のためのマシン語入門書と感じる人も多いかもしれません。ある意味ではそれも正しいのですが、実際にはゲームをネタにしてマシン語を理解するため、というのが本当の目的なのです。ゲームはあくまで素材でしかありません。

　では、なぜゲームをターゲットに選んだかといいますと、ゲームほどマシン語のテクニックを駆使しなければならないものはほかに見当たらないからです。マシン語が「ニモニックのパズル」である以上、用法をおぼえながらその成果（マシン語プログラム）が進歩していくというのは理想の姿です。本書にあるプログラムが、マシン語ゲーム制作の基礎編として活用されるだけでなく、純粋にマシン語の基礎として役に立つことができたなら、ゲームの世界に新しいジャンルができたといえるかもしれません。

5-1 消去ルーチン

パタパタ・アニメができるようになると、表示座標を変えていけばパターンが動くのではないか、ということに気がつかれた方もけっこう多いのではないでしょうか。

しかし、さらにもう一歩進んで考えてみると、そのまま座標だけを変えたのでは、前のパターンの残骸が残ってしまうということにも気がつくはずです。

■邪魔ものは消さねばならない

では、残骸をなくすにはどうしたらいいのでしょうか——これはとくに考えるほどの問題ではありませんね。残骸となっている古いパターンの一部を、新しいパターンを表示する前に消去しておけばいいのです。

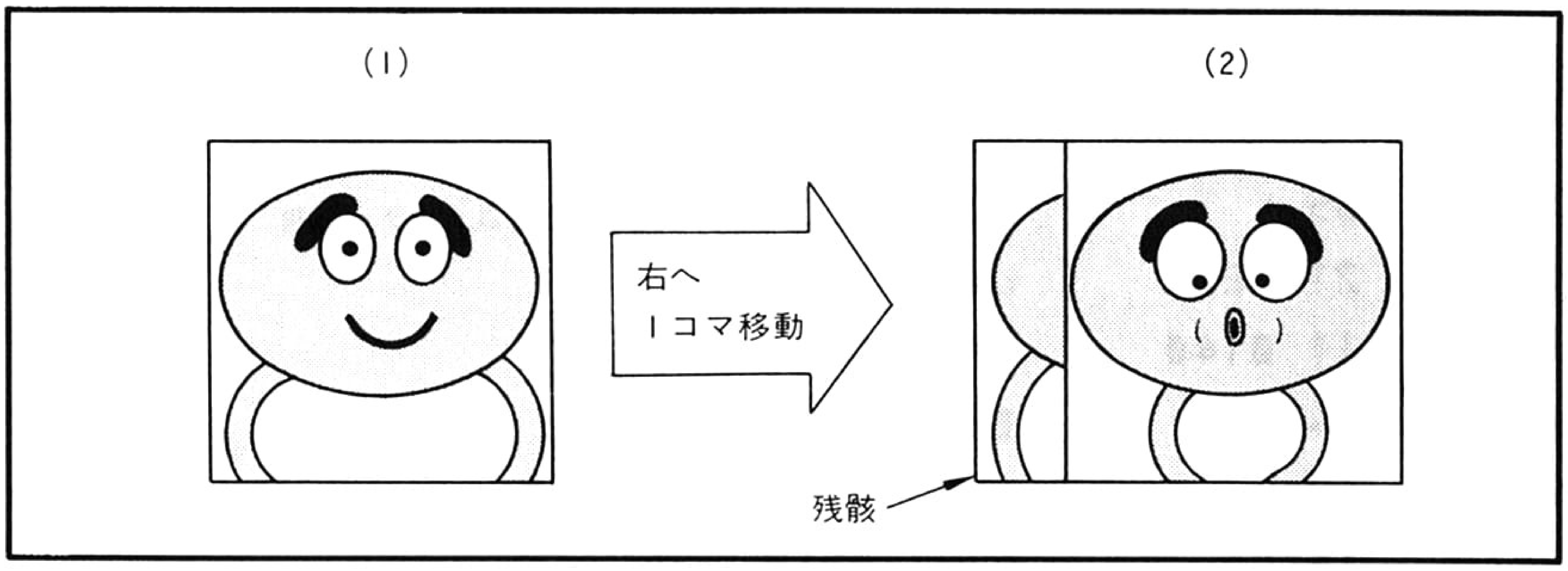


図5.1　パターン移動と残骸

画面に表示されているものを消すということは、そのグラフィックV.RAMに00_Hを入れるということですから、プログラム的には第3章で作成した正方形表示ルーチンと似たようなものになります。

たとえば図5.1のように、パターンを右へ1コマ移動して表示するのであれ

ば、移動表示する前に残骸のある四角形部分に、黒い四角形を描くということです。

■黒い四角形を描く

そこで、まず必要なのはこの黒い四角形表示ルーチンですが、ここでは単に一定サイズの四角形を描くものにするのではなく、いろいろと応用ができるように、サイズもつぎのように指定できるプログラムを作ることにします。

座　標＝（C，B）
サイズ＝Dバイト×Eドット

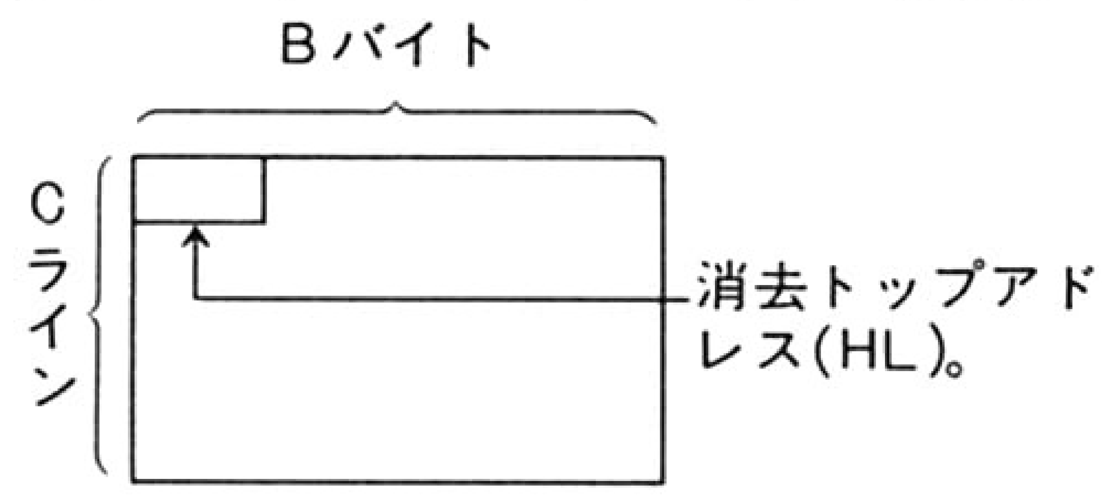
プログラムのアルゴリズムは基本的には正方形表示と同じですが、座標およびサイズがフリーになった分だけPUSH/POP命令が多いので、何を何のためにPUSHあるいはPOPしているのか、よく考えながらプログラムを組んでください。

また、このプログラム自体は単独でも動きますが、セーブする時は、第4章で作成したパターン表示ルーチンにアペンドしておくと便利です。

LIST 5-1

```
BD40   D5         PUSH D       DEの値をスタックへ退避。
BD41   21 BEC0    LXI  H,BEC0  ┐
BD44   11 0140    LXI  D,0140  │
BD47   04         INR  B       │
BD48   19         DAD  D       │ (C,B)よりHLレジスタに消去アドレスを
BD49   10 FD      DJNZ BD48    │ 求める。
BD4B   09         DAD  B       ┘
BD4C   11 0050    LXI  D,0050  DE← 0050H……次ラインへの増加バイト数。
BD4F   C1         POP  B       B……横の消去バイト数，C…縦の消去ライ
                               ン数となる。
BD50   AF         XRA  A       A←0……消去データ。
BD51   D3 5C      OUT  5C      ブルー面にバンク切り換え。
BD53   CD BD63    CALL BD63    四角形の消去。
```

BD56	D3 5D	OUT	5D	レッド面にバンク切り換え。
BD58	CD BD63	CALL	BD63	四角形の消去。
BD5B	D3 5E	OUT	5E	ブルー面にバンク切り換え。
BD5D	CD BD63	CALL	BD63	四角形の消去。
BD60	D3 5F	OUT	5F	メインRAMにバンク切り換え。
BD62	C9	RET		リターン。
BD63	C5	PUSH	B	消去サイズ,アドレスをスタックへ退避。
BD64	E5	PUSH	H	
BD65	C5	PUSH	B	HLよりB×Cの四角形を00Hで埋める。
BD66	E5	PUSH	H	
BD67	77	MOV	M,A	
BD68	23	INX	H	
BD69	10 FC	DJNZ	BD67	
BD6B	E1	POP	H	
BD6C	19	DAD	D	
BD6D	C1	POP	B	
BD6E	0D	DCR	C	
BD6F	20 F4	JRNZ	BD65	
BD71	E1	POP	H	消去サイズ,アドレスをスタックから取り出す。
BD72	C1	POP	B	
BD73	C9	RET		リターン。



LIST 5-2

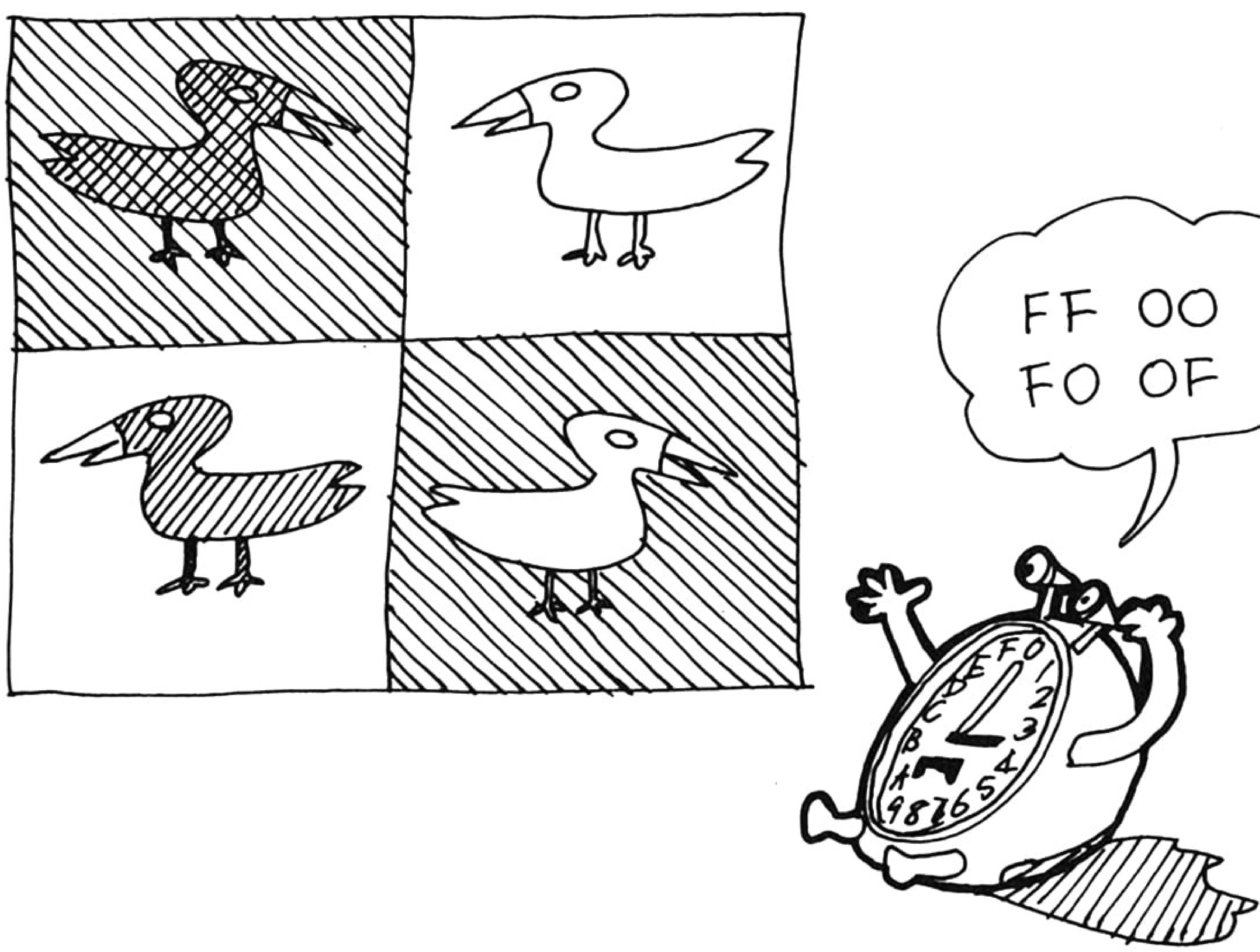
D000	F3	DI		割込み禁止。
D001	01 1010	LXI	B,1010	BC←1010H……消去座標＝(10H, 10H)。
D004	11 1040	LXI	D,1040	DE←1040H……消去サイズ＝10Hバイト×40Hライン。
D007	CD BD40	CALL	BD40	(C,B)よりサイズD×Eの四角形消去を行う。
D00A	FB	EI		割込み許可。
D00B	FF	RST	7	hコの状態へ戻る。

テスト・プログラムを実行する前に、画面を

```
LINE (0, 0)-(639, 199),7,BF
```

とでもして塗りつぶしておかないと、黒に黒では「闇夜のカラス」になってしまいます。

これもまた、例によって、いろいろな位置に、いろいろなサイズでテストをしてみてください。



5-2 移動方向別消去

あなたが現在マシン語のテスト用に使っているキャラクターパターンは、たぶん本書の中にあったものだと思いますが、これが自分のアイデアで作成したオリジナルパターンなると、動いた時の気分はまさに感動ものなのです。これは、マシン語がバグなしで動いてくれたということより、ピノキオを作ったゼペットじいさんの気持ちに近いものかもしれません。大げさにいえば、そこにはBASICでは味わうことができないマシン語の夢とロマンがあるといえます。

ここでは、そこまでの感動を味わうにはまだ遠いでしょうが、パターンが動くことにより、マシン語の勉強という暗い気分から少しでも抜けられれば、動いたパターンもきっと喜んでくれる（？）に違いありません。

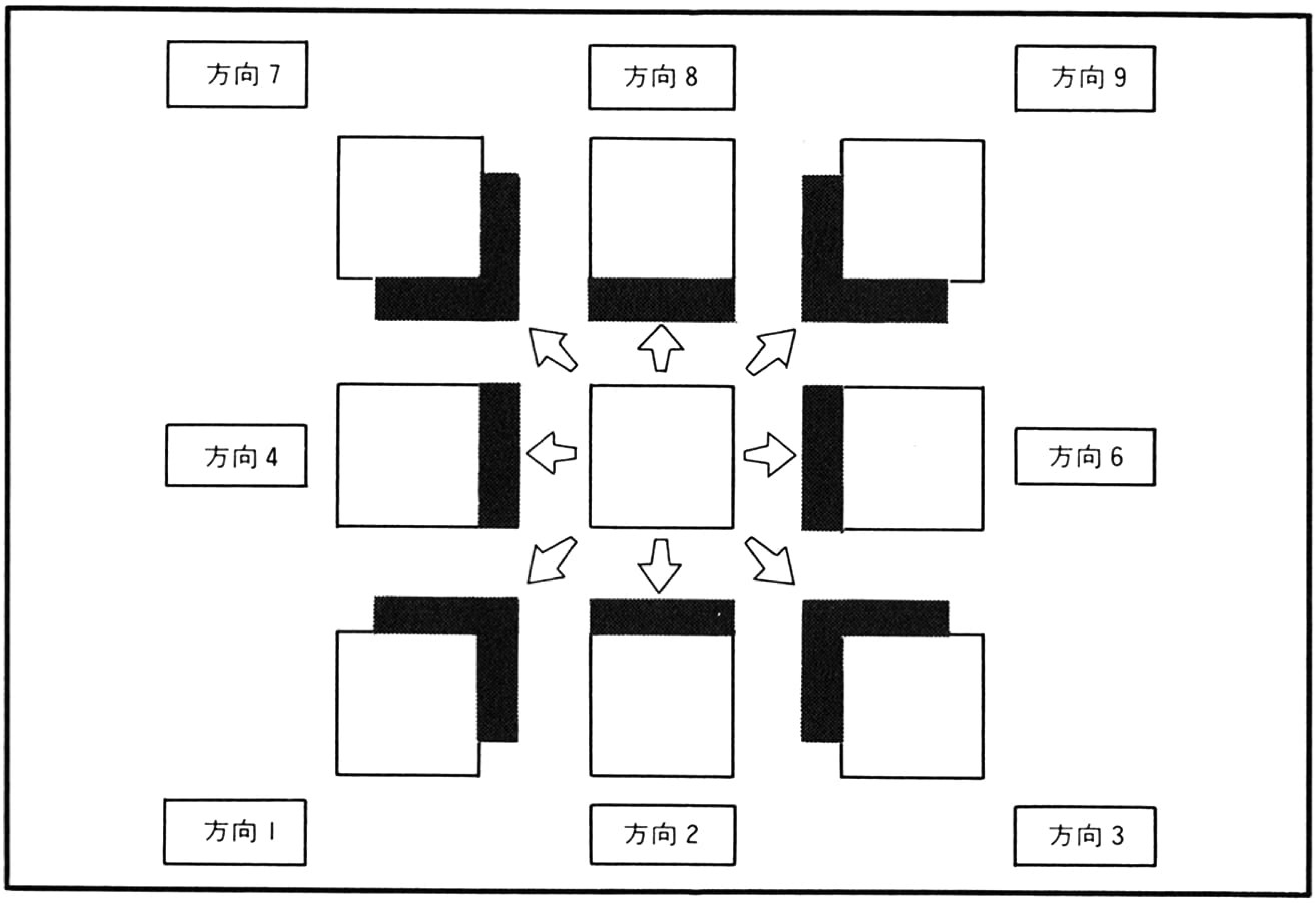
■パターンを上下左右に動かす

これまでに、すでにパターンを動かすための基本は理解しているし、そのための消去ルーチンも作成したわけですが、実際にパターンを動かすには、移動方向と消去の関係をもう少し具体的に結びつけたほうがよさそうです。

一般的に、移動する単位は座標1コマ分を基準としますから、方向と消去しなければならない部分の関係は図5.2のようになります。なお、方向番号は、わかりやすいようにテンキーの配列に合わせました。

本書が純粋にゲームを作るためのものであれば、全方向についての消去ルーチンを作るのが当然なのでしょうが、目的はあくまでもマシン語マスターですから、ここで作るのは縦・横の4方向についてだけにしてあります。斜めは一見面倒なようですが、L字形の黒い部分を2つの長方形に分けて消去すればいいので、自分でやってみてください（**LIST5-3**）。

図5.2 移動方向別の消去(黒い部分)



LIST 5-3

```
BD74    11 0404    LXI  D,0404   [方向2の消去]
BD77    C3 BD40    JMP  BD40     (C,B)より4バイト×4ラインの消去。
BD7A    0C         INR  C
BD7B    0C         INR  C        [方向4の消去]
BD7C    0C         INR  C        (C+3,B)より1バイト×16ラインの消去。
BD7D    11 0110    LXI  D,0110
BD80    C3 BD40    JMP  BD40
BD83    11 0110    LXI  D,0110   [方向6の消去]
BD86    C3 BD40    JMP  BD40     (C,B)より1バイト×16ラインの消去。
BD89    04         INR  B
BD8A    04         INR  B        [方向8の消去]
BD8B    04         INR  B        (C,B+3)より4バイト×4ラインの消去。
BD8C    11 0404    LXI  D,0404
BD8F    C3 BD40    JMP  BD40
```

■パターン移動作成にトライ

この方向別消去ルーチン（前ページ）は、パターンを表示した時の座標でコールするようになっていますから、ワークエリアにある座標をそのまま使用することができます。早速、これを利用してパターン移動にトライしてみましょう。

LIST 5-4

```
DØØØ   F3          DI          割込み禁止。
DØØ1   AF          XRA   A     A←0.
DØØ2   32 C5ØØ     STA   C5ØØ  (C500H)←A……… 初期X座標。
DØØ5   32 C5Ø1     STA   C5Ø1  (C501H)←A……… 初期Y座標。
DØØ8   32 C5Ø2     STA   C5Ø2  (C502H)←A……… 初期パターン番号。
DØØB   Ø6 4Ø       MVI   B,4Ø  B←40H………移動回数。
DØØD   C5          PUSH  B     BCの値をスタックへ退避。
DØØE   3A C5ØØ     LDA   C5ØØ  }
DØ11   4F          MOV   C,A   } C←(C500H)……X座標。
DØ12   3A C5Ø1     LDA   C5Ø1  }
DØ15   47          MOV   B,A   } B←(C501H)……Y座標。
DØ16   C5          PUSH  B     }
DØ17   CD BD83     CALL  BD83  } (C,B)より［方向6］へ移動するための部分
DØ1A   C1          POP   B     } を消去する。
DØ1B   ØC          INR   C     C←C＋1……移動後のX座標。
DØ1C   79          MOV   A,C   }
DØ1D   32 C5ØØ     STA   C5ØØ  } 移動後のX座標を(C500H)へ格納。
DØ2Ø   3A C5Ø2     LDA   C5Ø2  A←(C502H)……パターン番号。
DØ23   F5          PUSH  PSW   }
DØ24   CD BDØØ     CALL  BDØØ  } (C,B)にパターン番号Aを表示。
DØ27   F1          POP   PSW   }
DØ28   3C          INR   A     }
DØ29   E6 Ø1       ANI   Ø1    } パターン番号を0なら1に，1なら0にして
DØ2B   32 C5Ø2     STA   C5Ø2  } (C502H)へ格納。
```

```
D02E    0E A0       MVI  C,A0   ┐
D030    06 FF       MVI  B,FF   │
D032    10 FE       DJNZ D032   ├ ウエイト
D034    0D          DCR  C      │
D035    20 F9       JRNZ D030   ┘
D037    C1          POP  B      BCの値をスタックから出す。
D038    10 D3       DJNZ D00D   B(40H)回上記1コマ移動を繰り返す。
D03A    FB          EI          割込み許可。
D03B    FF          RST  7      h]の状態へ戻る。
```

テストに当たっては、パターンデータと、これまでに作成したメイン部のプログラムがロードされていることを確認してから実行してください。うまく動いて、そのうえ感動も味わえれば最高なのですが……。

■表示しては消す

さて、プログラム本体(**LIST5-4**)の内容ですが、ワークエリアの役割はパタパタ・アニメの時と全く同じで、$C500_H$番地から順にX座標・Y座標・パターン番号となっています。移動は方向6(右)に40_H(64)コマ、つまり長さにして512ドット分です。1コマずらして表示する前に、方向6の不要部分消去ルーチンをコールし、X座標を+1してからパターンを表示しています。

つぎに、パターン番号の0と1の切り換えですが、第4章の最後にあったように、現在のパターン番号を+1し、1とのANDを取って交互に0、1となるようにしています。

第5章といっても、これまで作成してきたプログラムはとくに新しい知識は必要としていないので、簡単過ぎて不満だったかもしれません。その辺は、マシン語への慣熟走行と思って耐えてください。

5-3 入力キーのチェック

マシン語ゲームの本ではない、などといいながら、内容はますますゲームに近づいていくようですが、今度はキャラクターパターンをテンキーによりコントロールできるようにしてみましょう。これは、すなわち**キースキャン**の方法ということになるわけですが、単に押されたキーを知りたいだけであれば、BASICのROM内ルーチン（35CE$_H$番地＝入力されたキーのアスキーコードをアキュムレータに入れてもどる）で十分かもしれません。

しかし本書は、ゲームを通じてのマシン語マスターが目標ですから、これでは何の役にも立たないも同然です。やはり、チェックしたいキーをROMなどに頼らずに調べられないと、たとえば裏RAMにプログラムを置くことができないとか、マシン語でありながらBASICの制限を受けることになってしまいます。

■誤操作のおそろしさ

テンキーによるパターン移動とは、一種のリモコン操作と同じことですから、押す側の人間が誤操作する可能性もあるわけです。つまり、画面の最下部にいるのにさらに下に行かせようとしたり、あるいはその反対のことをさせようとしたり、パターンに対して無理なことをさせようとするかもしれないということです。

もしも、高価なリモコン操作の飛行機に、こんな無理を強要したらどうなるでしょうか。アッという間に墜落するか、リモコンの効かない空の彼方へ飛んでいってしまうはずですね。

これは、画面上のパターンだって同じことです。画面から消え去るということは、プログラム実行結果からいえばグラフィックV.RAM以外のメモリにパターンデータを書き込むということですから、下手をすると、現在実行

中のマシン語プログラムの上に書くことだってあるわけです。こうなると、その瞬間にプログラムはメチャクチャになり、まず100パーセント間違いなく暴走することになります。

■ミスを防ぐ

このような危険を防ぐためには、操作する人間側に「無理はしないように」と納得してもらうか、プログラムのほうで無理なことに対する対策を、最初から立てておくことが必要です。

そうはいっても前者の方法では、プログラムを作成した本人以外は恐らくだれも納得しないでしょうから、これはプログラムで対処するしかありません。

では具体的にはどうすればいいのかというと、図5.3のように**画面エンド**を設けて、移動する前に移動可能位置かどうかエンドチェックをすればいいのです。

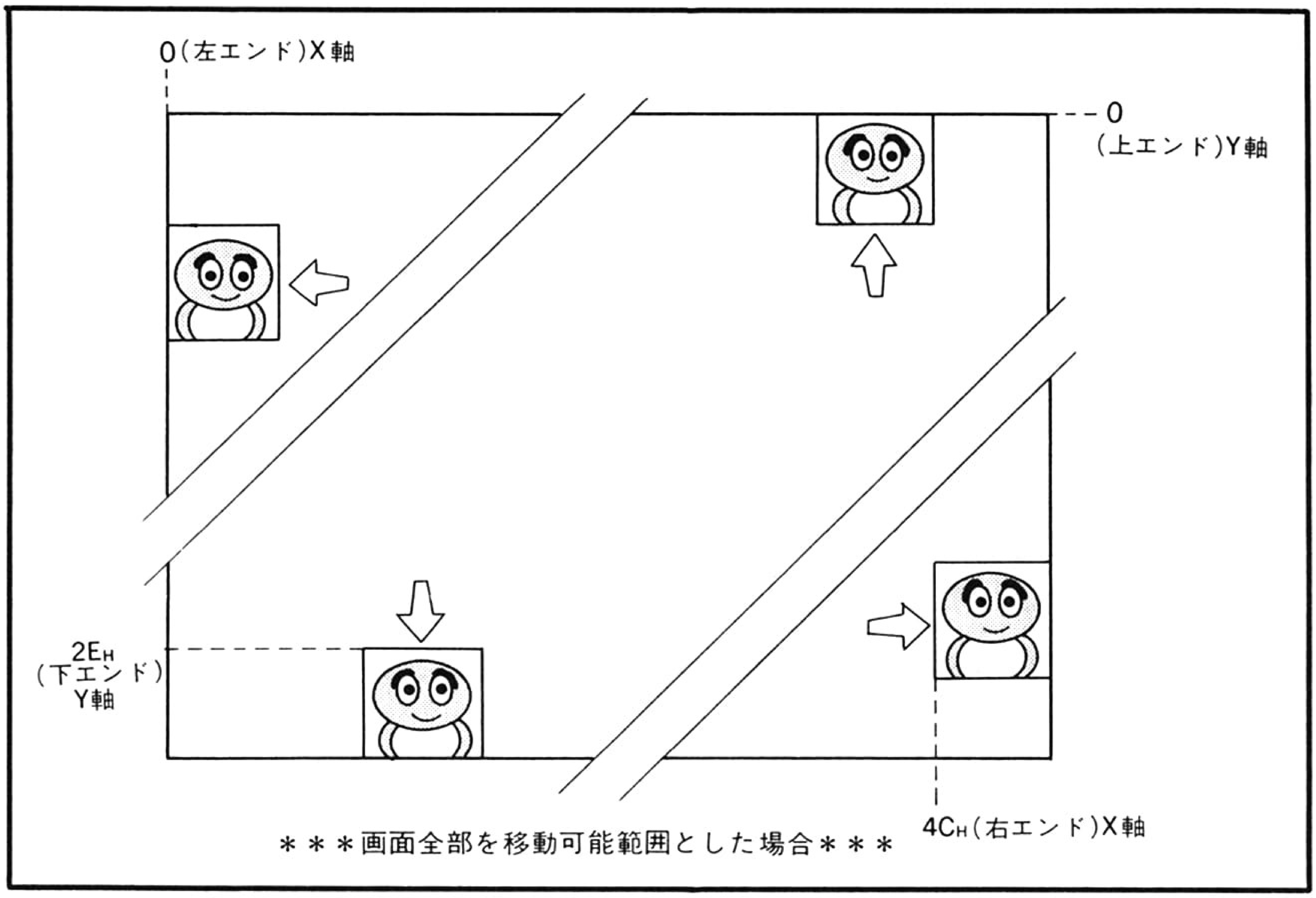


図5.3　移動パターンのエンド座標

なぜ、こんな簡単なことを長々と書いてあるか疑問に思うかもしれませんね。その理由は、キー入力に限らず、マシン語でプログラムを組む際には、例外的なことを例外と思わず、あらゆることを想定していかないと暴走につながる、ということを理解してもらいたかったからです。これは、エラーメッセージでエラー内容まで知らせてくれるBASICとの大きな違いです。

■エンドチェックのルーチン

というところで、各方向別にエンドチェックを行い、その結果により移動させる、あるいは移動させずに、パターンを表示するルーチンを作成することにしましょう。

LIST 5-5

```
BD92   3E 2E     MVI  A,2E    } B=2EH(下エンド)なら
BD94   B8        CMP  B       }   BDC4H番地(パターン表示)へジャンプ。
BD95   28 2D     JRZ  BDC4    }
BD97   C5        PUSH B       } (C,B)より[方向2]へ移動するための不要
BD98   CD BD74   CALL BD74    } 部分消去。
BD9B   C1        POP  B       }          &
BD9C   Ø4        INR  B       } B←B+1としてBDC4H番地へジャンプ。
BD9D   18 25     JMPR BDC4    }
BD9F   3E ØØ     MVI  A,ØØ    } C=0(左エンド)なら,
BDA1   B9        CMP  C       } BDC4H番地へジャンプ。
BDA2   28 2Ø     JRZ  BDC4    }
BDA4   C5        PUSH B       } (C,B)より[方向4]へ移動するための不要
BDA5   CD BD7A   CALL BD7A    } 部分消去
BDA8   C1        POP  B       }          &
BDA9   ØD        DCR  C       } C←C-1としてBDC4H番地へジャンプ
BDAA   18 18     JMPR BDC4    }
BDAC   3E 4C     MVI  A,4C    } C=4CH(右エンド)なら,
BDAE   B9        CMP  C       } BDC4H番地へジャンプ。
BDAF   28 13     JRZ  BDC4    }
```

BDB1	C5	PUSH B	(C,B)より［方向6］へ移動するための不要部分消去。 & C←C＋1としてBDC4H番地へジャンプ。
BDB2	CD BD83	CALL BD83	
BDB5	C1	POP B	
BDB6	0C	INR C	
BDB7	18 0B	JMPR BDC4	
BDB9	3E 00	MVI A,00	B＝0(上エンド)なら, BDC4H番地へジャンプ。
BDBB	B8	CMP B	
BDBC	28 06	JRZ BDC4	
BDBE	C5	PUSH B	(C,B)より［方向8］へ移動するための不要部分消去。 & B←B－1.
BDBF	CD BD89	CALL BD89	
BDC2	C1	POP B	
BDC3	05	DCR B	
BDC4	79	MOV A,C	(C,B)をそれぞれのワークエリア(C500H), (C501H)へ格納。
BDC5	32 C500	STA C500	
BDC8	78	MOV A,B	
BDC9	32 C501	STA C501	
BDCC	3A C502	LDA C502	A←(502H)……パターン番号。
BDCF	F5	PUSH PSW	(C,B)にパターン番号Aを表示。
BDD0	CD BD00	CALL BD00	
BDD3	F1	POP PSW	
BDD4	3C	INR A	パターン番号を0なら1に、1なら0にして, (C502H)へ格納。
BDD5	E6 01	ANI 01	
BDD7	32 C502	STA C502	
BDDA	C9	RET	リターン。

このLIST5-5で、本書におけるメイン部のプログラムは終わりです。$BD00_H$から通して１つのファイルにしてください。今回のものは、プログラムとしてはこれまでよりやや長めですが、実際には同じような内容のものが４方向分あるだけですから、とくに難しいことはないはずです。

■いよいよ完成

続けて、キー入力ルーチンをともなった**LIST5-6**を作成して、まずは実行（h] GD000 ⏎）によりプログラム結果を確認してみてください。これが、本書最後のテストプログラムとなります。

パターンがアニメのようにに動いてくれ、画面エンドではパタパタしているならば、うまくいった証拠です。

またテンキー移動だけだと、リセット以外にはプログラムを中止させることができませんが、ここではループの最後にストップキーのチェックを行い、ストップキーが押されているならばモニタ（h] の状態）へもどるようにしてあります。キー入力に関する部分を無視すれば、プログラムはコメントを見るだけで十分に理解できる内容ですから、早速キー入力の説明へと移ることにしましょう。

LIST 5-6

```
D000   F3         DI            割込み禁止。
D001   AF         XRA   A       A←0.
D002   32 C500    STA   C500    (C500H)←A………初期X座標。
D005   32 C501    STA   C501    (C501H)←A………初期Y座標。
D008   32 C502    STA   C502    (C502H)←A………初期パターン番号。
D00B   4F         MOV   C,A    ┐
D00C   47         MOV   B,A    ├ (0, 0)にパターン番号0を表示。
D00D   CD BD00    CALL  BD00   ┘
D010   3A C500    LDA   C500   ┐
D013   4F         MOV   C,A    ┘ C←X座標。
D014   3A C501    LDA   C501   ┐
D017   47         MOV   B,A    ┘ B←Y座標。
D018   DB 00      IN    00      A←入力ポート0Hの値。
D01A   07         RLC          ┐
D01B   07         RLC          ├ 6キーが押されていればD02EH番地へジャンプ
D01C   30 10      JRNC  D02E   ┘
```

DØ1E	Ø7	RLC	4キーが押されていればD033$_H$番地へジャンプ。
DØ1F	Ø7	RLC	
DØ2Ø	3Ø 11	JRNC DØ33	
DØ22	Ø7	RLC	2キーが押されていればD038$_H$番地へジャンプ。
DØ23	Ø7	RLC	
DØ24	3Ø 12	JRNC DØ38	
DØ26	DB Ø1	IN Ø1	A←入力ポート1$_H$の値。
DØ28	ØF	RRC	8キーが押されていれば,[方向8]へ1コマ移動ルーチンをコール。
DØ29	D4 BDB9	CNC BDB9	
DØ2C	18 ØD	JMPR DØ3B	D03B$_H$番地へジャンプ。
DØ2E	CD BDAC	CALL BDAC	[方向6]へ1コマ移動ルーチンをコールし,D03B$_H$番地へジャンプ。
DØ31	18 Ø8	JMPR DØ3B	
DØ33	CD BD9F	CALL BD9F	[方向4]へ1コマ移動ルーチンをコールし,D03B$_H$番地へジャンプ。
DØ36	18 Ø3	JMPR DØ3B	
DØ38	CD BD92	CALL BD92	[方向2]へ1コマ移動ルーチンをコール。
DØ3B	ØE AØ	MVI C,AØ	ウェイト。
DØ3D	Ø6 FF	MVI B,FF	
DØ3F	1Ø FE	DJNZ DØ3F	
DØ41	ØD	DCR C	
DØ42	2Ø F9	JRNZ DØ3D	
DØ44	DB Ø9	IN Ø9	A←入力ポート9$_H$の値。
DØ46	ØF	RRC	[STOP]キーが押されていなければD010$_H$番地へジャンプ。
DØ47	DA DØ1Ø	JC DØ1Ø	
DØ4A	FB	EI	割込み許可。
DØ4B	FF	RST 7	h]の状態へ戻る。

■キー入力ポート

キーの入力をパソコン本体側から見ると、テープレコーダーやライトペンなどと同じく、外部からのデータの入力ということになるわけですが、これら外部からの入力データを受け付ける本体側の窓口のことを**入力ポート**といいます。もちろん、1つの入力ポートですべてのデータを扱うことはできま

せんから、いくつかの入力ポートがあって、ポート別に扱うデータの範囲が決まっているわけです。

キーボードに関しては、入力ポートの00_H～$0E_H$が**図5-5**のように役割分担しており、押されたキーのビットは0、押されていないキーのビットは1になるようになっています。

（注）ポート番地OO_H～OB_HはSR/FR/MR以前の機種に共通。OC_H以降はFH/MH以降のもの。

■ローテート命令

入力ポートのデータは、IN命令によってアキュムレータに取り入れることができますから、スキャンしたいキーのビットが0か1かを調べればいいわけです。

LIST5-6では、特定のビットを調べるために、**ローテート命令**によりビットの値をキャリーフラグに入れて条件分岐させています。RLC（Rotate Left Circular）とRRC（Rotate Right Circular）の内容を図示すると、つぎのようになります。

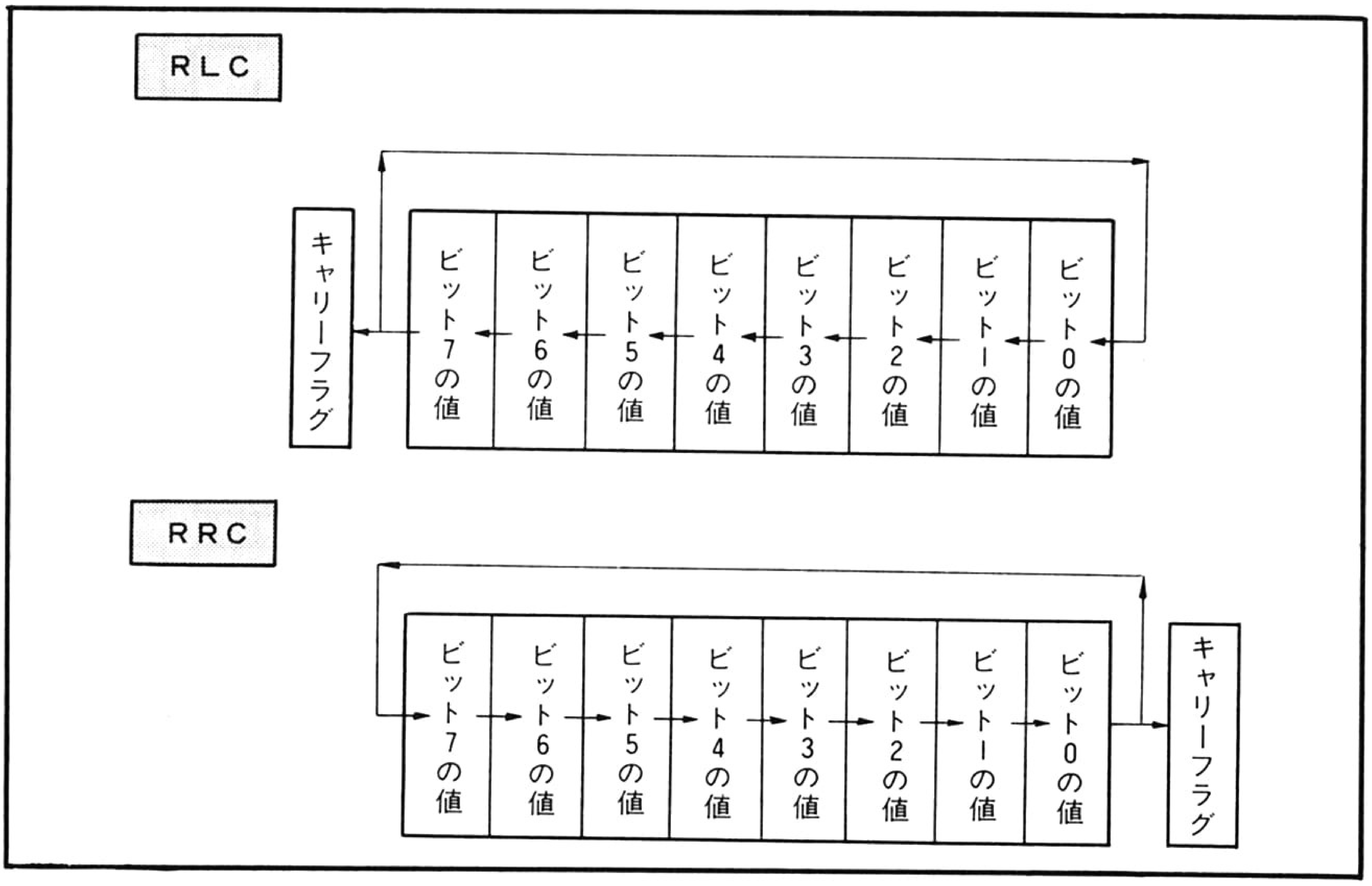


図5.4　RLCとRRCの内容

ポート＼ビット	7	6	5	4	3	2	1	0
00H	7	6	5	4	3	2	1	0
01H	RETURN	.	,	=	+	*	9	8
02H	G キ	F ハ	E イ ィ	D シ	C ソ	B コ	A チ	@ ~ ゛
03H	O ラ	N ミ	M モ	L リ	K ノ	J マ	I ニ	H ク
04H	W テ	V ヒ	U ナ	T カ	S ト	R ス	Q タ	P セ
05H	_ = ホ	^ へ	] } ム 」	¥ ¦ ー	[¦ 。 「	Z ッ ッ	Y ン	X サ
06H	7 ' ヤ ャ	6 & オ ォ	5 % エ ェ	4 $ ウ ゥ	3 # ア ァ	2 " フ	1 ! ヌ	0 ワ ヲ
07H	ー ロ	/ ? メ ・	. > ル 。	, < ネ 、	; + レ	: * ケ	9) ヨ ョ	8 (ユ ュ
08H	CTRL	SHIFT	カナ	GRPH	INS DEL	→	↑	HOME CLR
09H	ESC	SPACE	f·5	f·4	f·3	f·2	f·1	STOP
0AH	CAPS	/	−	COPY	HELP	←	↓	TAB
0BH	……	……	……	……	……	……	ROLL DOWN	ROLL UP
0CH	DEL	INS	BS	f·10	f·9	f·8	f·7	f·6
0DH	……	……	……	……	全角	PC	決定	変換
0EH	……	……	……	……	SHIFT 右	SHIFT 左	RETURN 10KEY	RETURN JIS側

図5.5　入力ポート別キーボードマップ

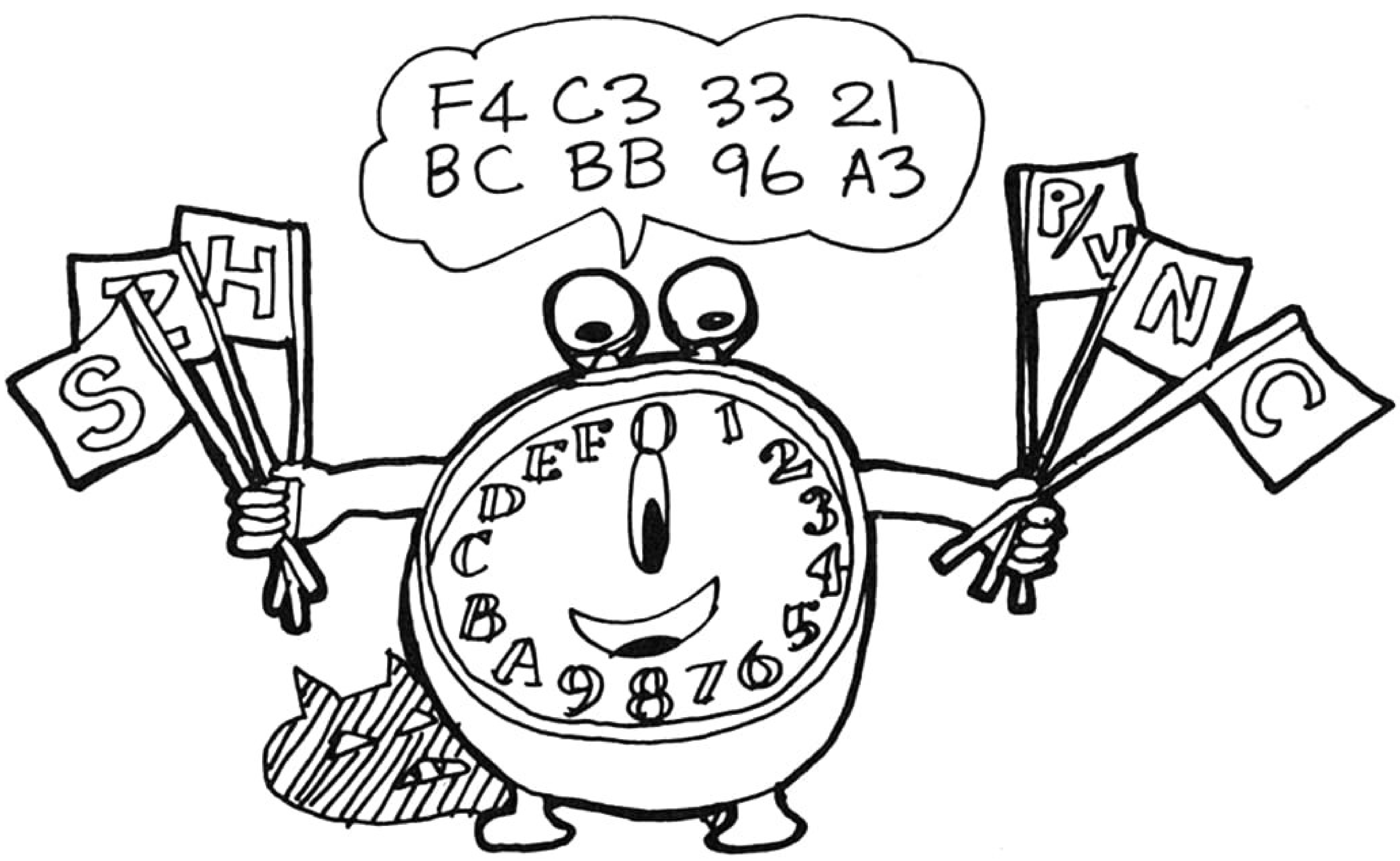
■キースキャンの実際

たとえば、6キーのチェックをしたいとすると、まず入力ポート00_Hの値をアキュムレータに入れ、RLCを2度行えばビット6の値がキャリーフラグに入るという具合です。

もしも、4キーだけをチェックしたければ、RLCを4回繰り返してもいいのですが、このような時は、例のAND命令を使うと1回ですむようになります。

```
IN    00   ………入力ポート 00Hの値をアキュムレータに入れる
ANI   10   ………10H（二進数で 0001 0000）とのANDを取る
JZ    xxxx………ゼロフラグが立っている（すなわちアキュムレータのビット
                    4＝0）ならxxxxH番地へジャンプ
```

この節では、キー入力によるパターンの移動をテーマにプログラムを作成してきましたが、じつはおぼえてほしいのは、このようなキャリーフラグの用法、そして論理演算の使い方なのです。そのうえで、プログラム細部への気配りができるようになると、もう「鬼に金棒」いや「XXさんにマシン語」と呼ばれるかもしれませんね。



5-4 モニタのメリット・デメリット

モニタ・アセンブラを使ってのマシン語プログラム作成、なれるとモニタもけっこう役に立つものです。ただし、ついその気になって大きなプログラムを作ろうなんて思うと、これは大変なことになってしまいます。恐らく、すぐに自信喪失、挫折のかたまりというマシン語拒否症にとりつかれること請け合いです。

それは、本書にあるようなリストをそのまま入力する時と違って、ジャンプやコール命令のアドレス計算、プログラムの訂正や追加挿入といった、プログラム作成上の基本的な作業環境があまりに悪いからです。

もちろん、どうしてもこのモニタで作ろうという超根性人間であれば、それも不可能というわけでもありません。実際に、このようなモニタ・アセンブラさえついていないPC－8801以外の機種では、今でも**ハンドアセンブル**という、プログラムを手作業で十六進数に変換する恐しい方法で、立派なプログラムを組む人もいらっしゃるというくらいです。

■Z-80とインテル8080

しかし、このような天才的例外人は基準にはなりえないわけで、一般的にはこのモニタ・アセンブラも使わないというのが現実です。そのうえ、これまでこのモニタ・アセンブラで使用してきたニモニックですが、じつはこのニモニックでは、本来PC－8801が持っている能力を全部引き出すことができないのです。

PC－8801は、ご存じのように**Z80**というザイログ社のCPUを搭載したコンピュータですが、このZ80はインテル社の**インテル8080**というCPUの改良版(バージョンアップ版)に当たるものです。そのためZ80は、インテル8080には含まれていなかった新しい内容の命令を持っています。

■Z-80とインテル8080のちがい

また、この2つのCPUは、同じ内容の命令はマシン語(十六進数)では同じ値になっていますが、ニモニックの表記法はかなり違っているのです。

ところがPC-8801のモニタは、どういうわけかインテル8080のニモニック(相対ジャンプ命令のみ例外)を採用しています。素直にザイログのニモニックを使用できるようになっていれば何も問題はないのですが、これが悩みの種となっています。

わかりやすいように、それを具体的に確認してみましょう。D000$_H$番地から、Sコマンドでつぎの値を入力してください。

```
ED 4B 00 C0 FF
```

これは、命令の内容としてはC000$_H$番地の中身をCレジスタに入れ、C001$_H$番地の中身をBレジスタに入れるという命令です。最後のFF$_H$はプログラム終了後h]の状態にもどるという意味ですから、最初の4バイトで1つの命令になっているわけです。

■ちがいの確認

ではこれを、Lコマンドにより**逆アセンブル**してみます。

```
h]LD000⏎
D000    ED          ???
D001    4B          MOV    C, E
D002    00          NOP
D003    C0          RNZ
D004    FF          RST    7
 :       :           :
```

この中で、インテル8080として正しいのは最後にあるRST7だけで、ほかは単に機械的にニモニックに直したか、それもできずに???などと意味不明のものになってしまっています。

とにかく、マシン語の命令としては正しいということを確認するために、BCレジスタの値を調べてから走らせてみてください。C001$_H$、C000$_H$番地の値が、それぞれBCレジスタに入っているはずです。

このように、Z80にはインテル8080に追加された命令がたくさんあるので、マシン語プログラムを組むうえで利用しない手はないというわけです。もちろん、そのような場合だけSコマンドで入力するということも可能ですが、そんな苦労などしなくても、じつは世の中にはもっと便利なものが用意されているのです。

■市販アセンブラ

このZ80ニモニックが使用できるだけでなく、最初に述べたモニタ・アセンブラの作業環境の悪さという欠点もなくしたのが、いわゆる市販アセンブラといわれているものです。これはPC－8801を買えばマシンに組み込まれているモニタと違い、ソフトとして別売されているものですから、利用しようと思えばお金を出して買わなければなりません。種類も『MF－ASM』や『DUAD－88D』などいろいろなものがあります。時にはパソコン雑誌などに個人が作成したアセンブラのプログラムが掲載されていることもありますが、入力が大変なうえ、それぞれで取り扱い方が違うため、この方法だとマシン語の入門者にとってはアセンブラ自体が学習上の最大のネットになってしまうともいえるくらいです。

■モニタ学習のメリット

本書が、モニタ・アセンブラを用いた第一の理由もここにあります。また、インテル8080ニモニックをおぼえることにもメリットがあります。単にモニタ・アセンブラが利用できるようになるだけでなく、将来、あなたが、たとえばCPUにインテル8086を使用しているPC－9801に進んだ場合に、ニモニックをおぼえる手間が少しは楽になるという面もあるのです。

ちなみに、さきほどのBCレジスタにC001$_H$、C000$_H$番地の中身を代入するという命令をザイログ形式により書いてみると、つぎのようになります。

```
LD　BC, (0C000H)
```

ザイログ形式では、数値の移動にはすべてこのようにLD（LoaDの略）命令を使うので、ニモニックとしてはインテル形式よりおぼえやすくなっています。

また、一般にアセンブラにおいては、0～9以外の数値で始まる十六進数については$0C000_H$というように、数値の先頭に0をつける決まりになっているので、これも発展性のある表記といえます。

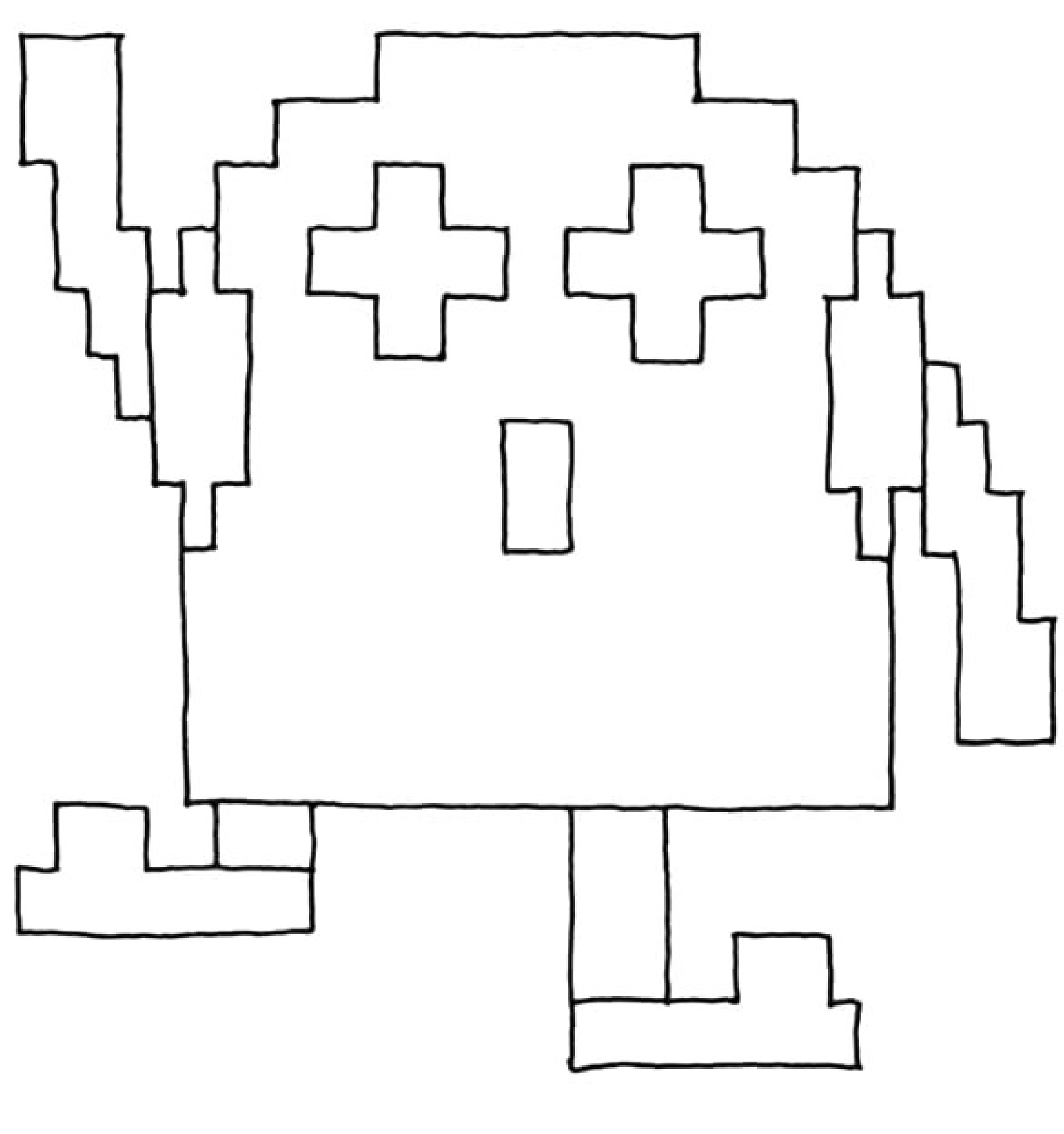
■まとめ

せっかくおぼえたモニタのインテル・ニモニックがあるのに、新たなニモニックをおぼえなければならないというのは、まわり道をした気がするかもしれませんが、ニモニックはマシン語プログラムになってはじめて意味を持つ言語ですから、その組み立て方法を知ることのほうがはるかに重要なのです。

ニモニック自体は、しょせんアルファベットですから、使い出せばすぐになれるものです。逆に、ザイログ形式からスタートしてしまうと、インテルの方式をおぼえるチャンスも必要もあまりないので、モニタを使えば簡単にできることでも、いちいちアセンブラに頼ることになりかねません。第一、モノになるかならないかわからないのに、高価なアセンブラを先に手に入れるなんて不経済な話ではありませんか。

本書では、とくにザイログ・ニモニックに関する説明はしていませんが、ニモニックは変化してもアルゴリズムやテクニックは同じです。今度はぜひ本格的な市販のアセンブラを用いて、本格的なマシン語プログラムに挑戦してみてください。もしも、ゲームに興味があるのであれば、拙著『マシン語ゲームプログラミング』（アスキー出版局）なんかは本書の続編として最高という気がします。

ちょっとCMが入りましたが、せっかくその気になったマシン語ですから、これで満足することなく、より奥深く追求されることを期待してやみません。



インテル8080ニモニック一覧表

［注1］ニモニックのオペランドで使われている記号

r　：8ビットレジスタ（A, B, C, D, E, H, L）

r&M　：r または M（HL レジスタで示されたアドレスの中身）

rr　：16ビットレジスタ（BC, DE, HL, SP）—— 表記上は B, D, H, SP となる

B&D　：BC または DE レジスタ —— 表記上は B, D となる

BDH　：BC, DE, HL レジスタ —— 表記上は B, D, H となる

n　：8ビットの数値（00_H～FF_H）—— 表記上は $_H$ は不要

p　：0～7の数値

nn　：16ビットの数値（0000_H～$FFFF_H$）—— 表記上は$_H$は不要

e　：相対アドレッシングの変位置 —— 現在アドレスを基準として-80_H～$+7F_H$ 番地の範囲に限られる

［注2］フラグ名

s　：サインフラグ

Z　：ゼロフラグ

H　：ハーフキャリーフラグ

P/V　：パリティ・オーバーフローフラグ

N　：減算フラグ

C　：キャリーフラグ

［注3］フラグ内の記号

.　：影響を受けない

0　：リセット

1　：セット

X　：不定

艘　：演算結果に従った影響を受ける

P　：“1” 偶数パリティ、“0” 奇数パリティ

V　：“1” オーバーフロー有り、“0” オーバーフロー無し

相対ジャンプ命令はインテル8080には本来ありませんが，PC-8801 のモニタでは使用できるようになっているため掲載してあります。

ニモニック	フラグ						Bite	内　　容
	S	Z	H	P/V	N	C		
ACI　n	↕	↕	↕	V	0	↕	2	Add with Carry Immediate ＝ アキュムレータに n と C フラグの値を加える
ADC　r&M	↕	↕	↕	V	0	↕	1	ADd with Carry ＝ アキュムレータに r&M で示される値と C フラグの値を加える
ADD　r&M	↕	↕	↕	V	0	↕	1	ADD ＝ アキュムレータに r&M で示される値を加える
ADI　n	↕	↕	↕	V	0	↕	2	ADd Immediate ＝ アキュムレータに n を加える
ANA　r&M	↕	↕	1	P	0	0	1	ANd A ＝ アキュムレータの値と r&M で示される値との AND を取る
ANI　n	↕	↕	1	P	0	0	2	ANd Immediate ＝ アキュムレータの値と n との AND を取る
CALL　nn	•	•	•	•	•	•	3	CALL ＝サブルーチン nn をコールする。次のアドレスが戻り番地としてスタックに入る
CC　nn	•	•	•	•	•	•	3	Call if Carry ＝ C フラグが立っていれば，サブルーチン nn をコールする
CM　nn	•	•	•	•	•	•	3	Call if Minus ＝ S フラグが立っていれば，サブルーチン nn をコールする
CMA	•	•	1	•	1	•	1	CoMplement Accumulator ＝ アキュムレータの各ビットの値を反転する
CMC	•	•	×	•	0	•	1	CoMplement Carry ＝ C フラグの値を反転する
CMP　r&M	↕	↕	↕	V	1	↕	1	CoMpare ＝ アキュムレータの値と r&M で示される値を比較（A-r&Mに相等）し，結果をフラグに反映させる
CNC　nn	•	•	•	•	•	•	3	Call if Non Carry ＝ C フラグが立っていなければ，サブルーチン nn をコールする
CNZ　nn	•	•	•	•	•	•	3	Call if Non Zero ＝ Z フラグが立っていなければ，サブルーチン nn をコールする
CP　nn	•	•	•	•	•	•	3	Call it Plus ＝ S フラグが立っていなければ，サブルーチン nn をコールする
CPE　nn	•	•	•	•	•	•	3	Call if Parity Even ＝ P フラグが立っていれば，サブルーチン nn をコールする

ニモニック	フラグ						Bite	内容
	S	Z	H	P/V	N	C		
CPI　n	↕	↕	↕	V	1	↕	2	ComPare Immediate = アキュムレータの値と n とを比較し，結果をフラグに反映させる
CPO　nn	•	•	•	•	•	•	3	Call if Parity Odd = P フラグが立っていなければ，サブルーチン nn をコールする
CZ　nn	•	•	•	•	•	•	3	Call if Zero = Z フラグが立っていれば，サブルーチン nn をコールする
DAA	↕	↕	↕	P	•	↕	1	Decimal Adjust Accumulator = BCD 演算の時に演算後のアキュムレータの値を補正する
DAD　rr	•	•	×	•	0	↕	1	Double ADd = HL レジスタに rr で示される値を加える
DCR　r&M	↕	↕	↕	V	1	•	1	DeCRment = r&M の値を −1 する
DCX　rr	•	•	•	•	•	•	1	DeCrement X (X：ペアレジスタ) = rr の値 −1 する
DI	•	•	•	•	•	•	1	Disable Interrupts = 割込み禁止
DJNZ　e	•	•	•	•	•	•	2	Decrement Jump if Non Zero = B = B−1 をして B = 0 でなければ e へジャンプする
EI	•	•	•	•	•	•	1	Enable Interrupts = 割込み許可
HLT	•	•	•	•	•	•	1	HaLT = プログラムの実行を中止する
In　n	•	•	•	•	•	•	2	INput = 入力ポート n の値をアキュムレータに取り入れる
INR　r&M	↕	↕	↕	V	0	•	1	INcRement = r&M の値を +1 する
INX　rr	•	•	•	•	•	•	1	INcrement X (X：ペアレジスタ) = rr の値を +1 する
JC　nn	•	•	•	•	•	•	3	Jump if Carry = C フラグが立っていれば nn へジャンプする
JM　nn	•	•	•	•	•	•	3	Jump if Minus = S フラグが立っていれば nn へジャンプする
JMP　nn	•	•	•	•	•	•	3	JuMP = nn へジャンプする
JMPR　e	•	•	•	•	•	•	2	JmMP Relative = e へジャンプする
JNC　nn	•	•	•	•	•	•	3	Jump if Non Carry = C フラグが立っていなければ nn へジャンプする

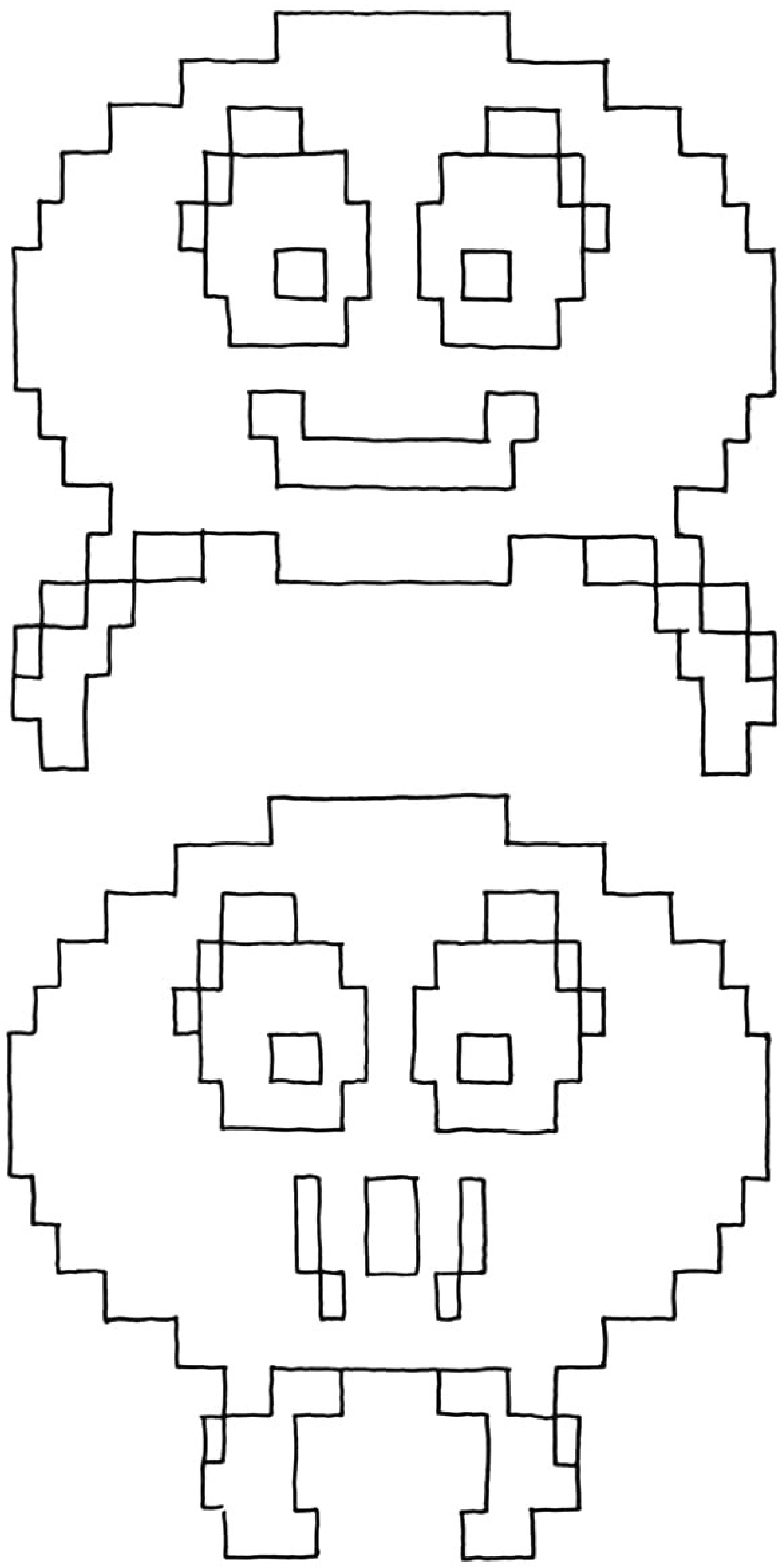
ニモニック	フ	ラ	グ				Bite	内　　　容
	S	Z	H	P/V	N	C		
JNZ　nn	•	•	•	•	•	•	3	Jump if Non Zero = Z フラグが立っていなければ nn へジャンプする
JP　nn	•	•	•	•	•	•	3	Jump if Plus = S フラグが立っていなければ nn へジャンプする
JPE　nn	•	•	•	•	•	•	3	Jump if Parlty Even = P フラグが立っていればnn へジャンプする
JPO　nn	•	•	•	•	•	•	3	Jump if Parity Odd = P フラグが立っていなければ nn へジャンプする
JRC　e	•	•	•	•	•	•	2	Jump Relative if Carry = C フラグが立っていれば e へジャンプする
JRNC　e	•	•	•	•	•	•	2	Jump Relative if Non Carry = C フラグが立っていなければ e へジャンプする
JRNZ　e	•	•	•	•	•	•	2	Jump Relative if non Zero = Z フラグが立っていなければ e へジャンプする
JRZ　e	•	•	•	•	•	•	2	Jump Relative if Zero = Z フラグが立っていれば e へジャンプする
JZ　nn	•	•	•	•	•	•	3	Jump if Zero = Z フラグが立っていれば nn へジャンプする
LDA　nn	•	•	•	•	•	•	3	LoaD Accumulator = アキュムレータに nn番地の中身を代入する
LDAX　B&D	•	•	•	•	•	•	1	LoaD Accumulator X = BC または DE レジスタで示される番地の中身をアキュムレータに入れる
LHLD　nn	•	•	•	•	•	•	3	Load HL Direct = nn+1 番地の中身を H に，nn番地の中身を L に代入する
LXI　rr, nn	•	•	•	•	•	•	3	Load X Immediate = rr に nn を代入する
MVI　r&M,n	•	•	•	•	•	•	2	MoVe Immediate = r&M に n を代入する
MOV　M, r	•	•	•	•	•	•	1	MOVe = HL レジスタで示された番地に r の値を入れる
MOV　r, M	•	•	•	•	•	•	1	MOVe = HL レジスタで示された番地の中身を r に代入する
MOV　r1, r2	•	•	•	•	•	•	1	MOVe = r2 の値を r1 に代入する

ニモニック	フラグ						Bite	内　　　容
	S	Z	H	P/V	N	C		
NOP	•	•	•	•	•	•	1	No OPeration = 何もしないで次の命令へ進む
ORA　r&M	↕	↕	0	P	0	0	1	OR Accumulator = アキュムレータの値と r&M で示される値との OR を取る
ORI　n	↕	↕	0	P	0	0	2	OR Immediate = アキュムレータの値と n とのOR を取る
OUT　n	•	•	•	•	•	•	2	OUTput = 出力ポート n にアキュムレータの値を出力する
PCHL	•	•	•	•	•	•	1	Program Counter is HL = HL レジスタの値をプログラム・カウンタとする。つまり HL で示される番地へジャンプすること
POP　BDH	•	•	•	•	•	•	1	POP = スタックの中身を BDH に取り出す
POP　PSW	↕	↕	↕	↕	↕	↕	1	POP Program Status Word = スタックの中身をアキュムレータとフラグに取り出す
PUSH　BDH	•	•	•	•	•	•	1	PUSH = BDH の値をスタックへ退避する
PUSH　PSW	•	•	•	•	•	•	1	PUSH Program Status Word = アキュムレータとフラグの値をスタックへ退避する
RAL	•	•	0	•	0	↕	1	Rotate Accumulator Left = C フラグをアキュムレータのビット 8 に見立て，一緒に左へローテーートする
RAR	•	•	0	•	0	↕	1	Rotate Accumulator Right = C フラグをアキュムレータのビット −1 に見立て，一緒に右へローデートする
RC	•	•	•	•	•	•	1	Return if Carry = C フラグが立っていればリターン
RET	•	•	•	•	•	•	1	RETurn = リターン（スタックの先頭 2 バイトへジャンプ）
RLC	•	•	0	•	0	↕	1	Rotate Left Circular = アキュムレータの中身を左にローテートする。ビット 7 の値は C フラグににも入る
RM	•	•	•	•	•	•	1	Return if Minus = S フラグが立っていればリターン

ニモニック	フラグ						Bite	内　　　容
	S	Z	H	P/V	N	C		
RNC	•	•	•	•	•	•	1	Return if Non Carry = C フラグが立っていなければリターン
RNZ	•	•	•	•	•	•	1	Return if Non Zero = Z フラグが立っていなければリターン
RP	•	•	•	•	•	•	1	Return if Plus = S フラグが立っていなければリターン
RPE	•	•	•	•	•	•	1	Return if Parity Even = P フラグが立っていればリターン
RPO	•	•	•	•	•	•	1	Return if Parity Odd = P フラグが立っていなければリターン
RRC	•	•	0	•	0	↕	1	Rotato Right Cir¢ular = アキュムレータの値を右にローテートする。ビット 0 の値は C フラグにも入る
RST　p	•	•	•	•	•	•	1	ReSTart = p x 8 番地をサブルーチンとしてコールする。p=7 の時 h] [redacted] の状態に戻る
RZ	•	•	•	•	•	•	1	Return if Zero = Z フラグが立っていればリターン
SBB　r&M	↕	↕	↕	V	1	↕	1	SuBtract with Borrow = アキュムレータから r&M で示される値と C フラグの値を引く
SBI　n	↕	↕	↕	V	1	↕	2	SuBtract Immediate = アキュムレータから n を引く
SHLD　nn	•	•	•	•	•	•	3	Store HL Directe = nn 番地に L を，nn + l 番地に H の値をそれぞれ入れる
SPHL	•	•	•	•	•	•	1	Stack Pointer is HL = HL レジスタの値を SP に代入する
STA　nn	•	•	•	•	•	•	3	Store Accumulator = アキュムレータの値を nn 番地に入れる
STAX　B&D	•	•	•	•	•	•	1	STore Accumulator X = アキュムレータの値を B&D で示される番地に入る
STC	•	•	0	•	0	1	1	SeT Carry = C フラグを立てる（1 にする）
SUB　r&M	↕	↕	↕	V	1	↕	1	SUBtract = アキュムレータから r&M で示される値を引く

ニモニック	フラグ						Bite	内　　容
	S	Z	H	P/V	N	C		
SUI　n	↕	↕	↕	V	1	↕	2	SUbtract Immediate = アキュムレータから n を引く
XCHG	•	•	•	•	•	•	1	eXCHanGe = DE レジスタと HL レジスタの値を交換する
XRA　r&M	↕	↕	0	P	0	0	1	XoR Accumulator = アキュムレータの値と r&M との XOR を取る
XRI　n	↕	↕	0	P	0	0	2	XOR Immediate = アキュムレータの値と n との XOR を取る
XTHL	•	•	•	•	•	•	1	eXchange stack with HL = スタックの先頭 2 バイトの値と HL レジスタの値を交換する

おまけの章
ザイログ・ニモニックへのステップ



この章では、ザイログ・ニモニックの世界への簡単な道案内をしたいと思います。

とはいいましても、ページに限りのある本である以上、これまでのようなくわしい説明はとてもできません。そこで、ここまでマシン語をマスターした皆さんなら、あとは自分の力で努力をしてくれるという前提で、第4章以降のマシン語プログラムの主要部分を、Z80とインテル8080の両方のニモニックで記述したリストを掲載することにしました。あわせて、Z80とインテル8080ニモニック対照表、そしてZ80ニモニック一覧表も掲載しておきますので、Z80ニモニックのスタディ用として利用してください。

ここにあるZ80ニモニックによるプログラムは、市販のアセンブラがないことには実際に入力することは不可能ですが、とりあえずZ80ニモニックの書式はおぼえられるので、アセンブラを手に入れるまでの下準備と思って、両者の違いだけでも研究しておくのもいいかもしれません。

一般にどのアセンブラでも、ソースリスト（アセンブルする前のニモニックリスト）の中に「；」を付けることにより、プログラムに関係のないコメントを入れることができるようになっています。ここでのインテル・ニモニックも、それを利用してコメントとしてアセンブラのソースリストに書き加えたものです。また、ラベルが使えるということや、数値に十六進数だけでなく十進数や足し算・引き算などの式までもが使用できるなど、モニタと比較していかに便利であるかということも、身をもって感じられると思います。

なお、PC-8800シリーズで使用できるZ80用アセンブラは、これまでに各社からいろいろなタイプのものが発売になっていますが、現在ではほとんどが市場から姿を消してしまいました。また、価格も安価なもので5,000円以上、高いものになると数万円というのが実情です。

そこで、次のステップのために、筆者自らが開発した本格的スクリーン・エディタ付きアセンブラ『HIT-88』(VER. 1.2) を推薦しておきます。

入手ご希望の方は、現金書留（または郵便小為替＋簡易書留）にて2,000円を下記へお送りください。申し込みの有効期限は、巻末に掲載されている本書最終発行日より1年間です。それ以降は、往復ハガキにてご確認ください。

〒353 埼玉県志木市館1-6-14-103
新日本プログラミング㈲／日高徹

『HIT-88』(VER. 1.2)は5" 2Dディスク専用タイプで、PC-8800全シリーズに対応しています。強力なエディタと高速アセンブラが、快適なマシン語開発環境を提供してくれます。BASICとはリセットなしで行き来でき、しかも市販ソフトのような長大なプログラムも作成可能ですから、初心者からプロまで安心して使うことができます。ぜひ、マシン語であなたの夢とロマンを実現してください。

```
;;;;;;;;;;;;;;;;;;;;;;;;;
;                       ;
;       ダ゛イ 4 ショウ       ;
;                       ;
;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;; LIST 4-1 ;;;;;;;

        ORG     0BD00H

DISP:   LD      HL,0BEC0H       ; LXI  H,BEC0
        LD      DE,320          ; LXI  D,0140
        INC     B               ; INR  B
XYLP:   ADD     HL,DE           ; DAD  D
        DJNZ    XYLP            ; DJNZ BD07
        ADD     HL,BC           ; DAD  B
        EX      DE,HL           ; XCHG

;;;;;;; LIST 4-2 ;;;;;;;

        LD      HL,0B740H       ; LXI  H,B740
        LD      BC,0C0H         ; LXI  B,00C0
        INC     A               ; INR  A
PTLP:   ADD     HL,BC           ; DAD  B
        DEC     A               ; DCR  A
        JR      NZ,PTLP         ; JRNZ BD13

;;;;;;; LIST 4-3 ;;;;;;;

        OUT     (5CH),A         ; OUT  5C
        CALL    BOX             ; CALL BD29
        OUT     (5DH),A         ; OUT  5D
        CALL    BOX             ; CALL BD29
        OUT     (5EH),A         ; OUT  5E
        CALL    BOX             ; CALL BD29
        OUT     (5FH),A         ; OUT  5F
        RET                     ; RET

BOX:    PUSH    DE              ; PUSH D
        LD      C,16            ; MCI  C,10
BLP1:   LD      B,4             ; MVI  B,04
BLP2:   LD      A,(HL)          ; MOV  A,M
        LD      (DE),A          ; STAX D
        INC     HL              ; INX  H
        INC     DE              ; INX  D
        DJNZ    BLP2            ; DJNZ BD2E
        PUSH    HL              ; PUSH H
```

```
        LD      HL,4CH          ; LXI  H,004C
        ADD     HL,DE           ; DAD  D
        EX      DE,HL           ; XCHG
        POP     HL              ; POP  H
        DEC     C               ; DCR  C
        JR      NZ,BLP1         ; JRNZ BD2C
        POP     DE              ; POP  D
        RET                     ; RET

;;;;;;; LIST 4-6 ;;;;;;;

        ORG     0D000H

        DI                      ; DI
        LD      A,0             ; MVI  A,00
        LD      (0C500H),A      ; STA  C500
        LD      (0C501H),A      ; STA  C501
        LD      (0C502H),A      ; STA  C502
        LD      B,50H           ; MVI  B,50

LOOP:   PUSH    BC              ; PUSH B
        LD      A,(0C500H)      ; LDA  C500
        LD      C,A             ; MOV  C,A
        LD      A,(0C501H)      ; LDA  C501
        LD      B,A             ; MOV  B,A
        LD      A,(0C502H)      ; LDA  C502
        PUSH    AF              ; PUSH PSW
        CALL    DISP            ; CALL BD00
        POP     AF              ; POP  PSW
        CP      1               ; CPI  01
        JR      Z,PTDEC         ; JRZ  D026
        INC     A               ; INR  A
        JR      PTST            ; JMPR D027
PTDEC:  DEC     A               ; DCR  A
PTST:   LD      (0C502H),A      ; STA  C502
        LD      C,0B0H          ; MVI  C,B0
PL1:    LD      B,0FFH          ; MVI  B,FF
        DJNZ    $               ; DJNZ D02E
        DEC     C               ; DCR  C
        JR      NZ,PL1          ; JRNZ D02C
        POP     BC              ; POP  B
        DJNZ    LOOP            ; DJNZ D00E
        EI                      ; EI
        RST     38H             ; RST  7
```

```
;;;;;;;;;;;;;;;;;;;;;;;;
;                      ;
;       P.EDITOR       ;
;                      ;
;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;; List 4-7 ;;;;;;;

        ORG     0BF00H

        JP      MDATA           ; JMP   BF09
        JP      DISPA           ; JMP   BF53

SIZE:   DW      410H            ; DJNZ  BF0C
PTNO:   DB      0               ; NOP

MDATA:  DI                      ; DI
        LD      HL,0B800H-0C0H  ; LXI   H,B740
        LD      DE,0C0H         ; LXI   D,00C0
        LD      A,(PTNO)        ; LDA   BF08
        INC     A               ; INR   A
        LD      B,A             ; MOV   B,A
MDLP1:  ADD     HL,DE           ; DAD   D
        DJNZ    MDLP1           ; DJNZ  BF15
        PUSH    HL              ; PUSH  H
        LD      B,A             ; MOV   B,A
        LD      HL,0C000H-780H  ; LXI   H,B880
        LD      DE,780H         ; LXI   D,0780
MDLP2:  ADD     HL,DE           ; DAD   D
        DJNZ    MDLP2           ; DJNZ  BF20
        POP     DE              ; POP   D
        OUT     (5CH),A         ; OUT   5C
        CALL    MBOX            ; CALL  BF37
        OUT     (5DH),A         ; OUT   5D
        CALL    MBOX            ; CALL  BF37
        OUT     (5EH),A         ; OUT   5E
        CALL    MBOX            ; CALL  BF37
        OUT     (5FH),A         ; OUT   5F
        EI                      ; EI
        RET                     ; RET

MBOX:   PUSH    HL              ; PUSH  H
        LD      A,(SIZE)        ; LDA   BF06
        LD      C,A             ; MOV   C,A
MBLP1:  LD      A,(SIZE+1)      ; LDA   BF07
        LD      B,A             ; MOV   B,A
        PUSH    HL              ; PUSH  H
```

```
MBLP2:  LD      A,(HL)          ;  MOV  A,M
        LD      (DE),A          ;  STAX D
        INC     HL              ;  INX  H
        INC     DE              ;  INX  D
        DJNZ    MBLP2           ;  DJNZ BF41
        POP     HL              ;  POP  H
        PUSH    DE              ;  PUSH D
        LD      DE,80           ;  LXI  D,0050
        ADD     HL,DE           ;  DAD  D
        POP     DE              ;  POP  D
        DEC     C               ;  DCR  C
        JR      NZ,MBLP1        ;  JRNZ BF3C
        POP     HL              ;  POP  H
        RET                     ;  RET

DISPA:  DI                      ;  DI
        XOR     A               ;  XRA  A
DISL1:  PUSH    AF              ;  PUSH PSW
        CALL    DISP            ;  CALL BF61
        POP     AF              ;  POP  PSW
        INC     A               ;  INR  A
        CP      6               ;  CPI  06
        JR      NZ,DISL1        ;  JRNZ BF55
        EI                      ;  EI
        RET                     ;  RET

DISP:   INC     A               ;  INR  A
        LD      B,A             ;  MOV  B,A
        LD      HL,0C000H-780H  ;  LXI  H,B880
        LD      DE,780H         ;  LXI  D,0780
DPLP1:  ADD     HL,DE           ;  DAD  D
        DJNZ    DPLP1           ;  DJNZ BF69
        PUSH    HL              ;  PUSH H
        LD      B,A             ;  MOV  B,A
        LD      HL,0B800H-0C0H  ;  LXI  H,B740
        LD      DE,0C0H         ;  LXI  D,00C0
DPLP2:  ADD     HL,DE           ;  DAD  D
        DJNZ    DPLP2           ;  DJNZ BF74
        POP     DE              ;  POP  D
        OUT     (5CH),A         ;  OUT  5C
        CALL    DBOX            ;  CALL BF8A
        OUT     (5DH),A         ;  OUT  5D
        CALL    DBOX            ;  CALL BF8A
        OUT     (5EH),A         ;  OUT  5E
        CALL    DBOX            ;  CALL BF8A
        OUT     (5FH),A         ;  OUT  5F
        RET                     ;  RET
```

```
DBOX:   PUSH    DE              ;   PUSH D
        LD      A,(SIZE)        ;   LDA  BF06
        LD      C,A             ;   MOV  C,A
DBL1:   LD      A,(SIZE+1)      ;   LDA  BF07
        LD      B,A             ;   MOV  B,A
        PUSH    DE              ;   PUSH D
DBL2:   LD      A,(HL)          ;   MOV  A,M
        LD      (DE),A          ;   STAX D
        INC     HL              ;   INX  H
        INC     DE              ;   INX  D
        DJNZ    DBL2            ;   DJNZ BF94
        POP     DE              ;   POP  D
        PUSH    HL              ;   PUSH H
        LD      HL,80           ;   LXI  H,0050
        ADD     HL,DE           ;   DAD  D
        EX      DE,HL           ;   XCHG
        POP     HL              ;   POP  H
        DEC     C               ;   DCR  C
        JR      NZ,DBL1         ;   JRNZ BF8F
        POP     DE              ;   POP  D
        RET                     ;   RET
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;
;                       ;
;       ﾀﾞｲ 5 ｼｮｳ       ;
;                       ;
;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;; List 5-1 ;;;;;;;

        ORG     0BD40H

DISP:   EQU     0BD00H

DCLS:   PUSH    DE              ;  PUSH D
        LD      HL,0BEC0H       ;  LXI  H,BEC0
        LD      DE,320          ;  LXI  D,0140
        INC     B               ;  INR  B
DCLP:   ADD     HL,DE           ;  DAD  D
        DJNZ    DCLP            ;  DJNZ BD48
        ADD     HL,BC           ;  DAD  B
        LD      DE,80           ;  LXI  D,0050
        POP     BC              ;  POP  B
        XOR     A               ;  XRA  A
        OUT     (5CH),A         ;  OUT  5C
        CALL    CBOX            ;  CALL BD63
        OUT     (5DH),A         ;  OUT  5D
        CALL    CBOX            ;  CALL BD63
        OUT     (5EH),A         ;  OUT  5E
        CALL    CBOX            ;  CALL BD63
        OUT     (5FH),A         ;  OUT  5F
        RET                     ;  RET

CBOX:   PUSH    BC              ;  PUSH B
        PUSH    HL              ;  PUSH H
DLP1:   PUSH    BC              ;  PUSH B
        PUSH    HL              ;  PUSH H
DLP2:   LD      (HL),A          ;  MOV  M,A
        INC     HL              ;  INX  H
        DJNZ    DLP2            ;  DJNZ BD67
        POP     HL              ;  POP  H
        ADD     HL,DE           ;  DAD  D
        POP     BC              ;  POP  B
        DEC     C               ;  DCR  C
        JR      NZ,DLP1         ;  JRNZ BD65
        POP     HL              ;  POP  H
        POP     BC              ;  POP  B
        RET                     ;  RET
```

```
;;;;;;; LIST 5-3 ;;;;;;;

D2CL:   LD      DE,0404H        ; LXI  D,0404
        JP      DCLS            ; JMP  BD40

D4CL:   INC     C               ; INR  C
        INC     C               ; INR  C
        INC     C               ; INR  C
        LD      DE,0110H        ; LXI  D,0110
        JP      DCLS            ; JMP  BD40

D6CL:   LD      DE,0110H        ; LXI  D,0110
        JP      DCLS            ; JMP  BD40

D8CL:   INC     B               ; INR  B
        INC     B               ; INR  B
        INC     B               ; INR  B
        LD      DE,0404H        ; LXI  D,0404
        JP      DCLS            ; JMP  BD40

;;;;;;; LIST 5-5 ;;;;;;;

MOV2:   LD      A,46            ; MVI  A,2E
        CP      B               ; CMP  B
        JR      Z,SAMP          ; JRZ  BDC4
        PUSH    BC              ; PUSH B
        CALL    D2CL            ; CALL BD74
        POP     BC              ; POP  B
        INC     B               ; INR  B
        JR      SAMP            ; JMPR BDC4

MOV4:   LD      A,0             ; MVI  A,00
        CP      C               ; CMP  C
        JR      Z,SAMP          ; JRZ  BDC4
        PUSH    BC              ; PUSH B
        CALL    D4CL            ; CALL BD7A
        POP     BC              ; POP  B
        DEC     C               ; DCR  C
        JR      SAMP            ; JMPR BDC4

MOV6:   LD      A,76            ; MVI  A,4C
        CP      C               ; CMP  C
        JR      Z,SAMP          ; JRZ  BDC4
        PUSH    BC              ; PUSH B
        CALL    D6CL            ; CALL BD83
        POP     BC              ; POP  B
        INC     C               ; INR  C
        JR      SAMP            ; JMPR BDC4
```

```
MOV8:   LD      A,0             ; MVI  A,00
        CP      B               ; CMP  B
        JR      Z,SAMP          ; JRZ  BDC4
        PUSH    BC              ; PUSH B
        CALL    D8CL            ; CALL BD89
        POP     BC              ; POP  B
        DEC     B               ; DCR  B

SAMP:   LD      A,C             ; MOV  A,C
        LD      (0C500H),A      ; STA  C500
        LD      A,B             ; MOV  A,B
        LD      (0C501H),A      ; STA  C501
        LD      A,(0C502H)      ; LDA  C502
        PUSH    AF              ; PUSH PSW
        CALL    DISP            ; CALL BD00
        POP     AF              ; POP  PSW
        INC     A               ; INR  A
        AND     1               ; ANI  01
        LD      (0C502H),A      ; STA  C502
        RET                     ; RET


;;;;;;; LIST 5-6 ;;;;;;;

        ORG     0D000H

        DI                      ; DI
        XOR     A               ; XRA  A
        LD      (0C500H),A      ; STA  C500
        LD      (0C501H),A      ; STA  C501
        LD      (0C502H),A      ; STA  C502
        LD      C,A             ; MOV  C,A
        LD      B,A             ; MOV  B,A
        CALL    DISP            ; CALL BD00

TLOOP:  LD      A,(0C500H)      ; LDA  C500
        LD      C,A             ; MOV  C,A
        LD      A,(0C501H)      ; LDA  C501
        LD      B,A             ; MOV  B,A
        IN      A,(0)           ; IN   00
        RLCA                    ; RLC
        RLCA                    ; RLC
        JR      NC,MD6          ; JRNC D02E
        RLCA                    ; RLC
        RLCA                    ; RLC
```

```
        JR      NC,MD4          ;  JRNC D033
        RLCA                    ;  RLC
        RLCA                    ;  RLC
        JR      NC,MD2          ;  JRNC D038
        IN      A,(1)           ;  IN   01
        RRCA                    ;  RRC
        CALL    NC,MOV8         ;  CNC  BDB9
        JR      TIME            ;  JMPR D03B

MD6:    CALL    MOV6            ;  CALL BDAC
        JR      TIME            ;  JMPR D03B

MD4:    CALL    MOV4            ;  CALL BD9F
        JR      TIME            ;  JMPR D03B

MD2:    CALL    MOV2            ;  CALL BD92
TIME:   LD      C,0A0H          ;  MVI  C,A0
TSL:    LD      B,0FFH          ;  MVI  B,FF
        DJNZ    $               ;  DJNZ D03F
        DEC     C               ;  DCR  C
        JR      NZ,TSL          ;  JRNZ D03D
        IN      A,(9)           ;  IN   09
        RRCA                    ;  RRC
        JP      C,TLOOP         ;  JC   D010
        EI                      ;  EI
        RST     38H             ;  RST  7
```

マシン語インストラクション一覧表

次ページ以降に掲載するマシン語インストラクション一覧表は、日本電気株式会社の「μCOM－82インストラクション活用表」より、転載いたしました。

μCOM－82とは、Z80をベースにしてPC－8801用に開発された特製のCPU（チップとかマイクロプロセッサともいう）の名称です。

〔注1〕 ニモニック中の略号の意味

略　号	意　　　味
d, n	8ビット・イミーディエート・データ
e	相対アドレシングの変位置
e−2	eの実効変位置

〔注2〕 フラグの記号の意味

記　号		意　　　味
•		影響受けない
0		リセット
1		セット
×		不定
↕		演算結果に従った影響受ける(①②③あり＝下に説明)
P		1＝偶数パリティ，0＝奇数パリティ
V		1＝オーバーフローあり，0＝オーバーフローなし
IFF		P/V フラグ ← (IFF)
↕	①	BC−1＝0 ならば P/V＝0，その他 P/V＝1
	②	A＝(HL) ならば Z＝1，その他 Z＝0
	③	B−1＝0 ならば Z＝1，その他 Z＝0

〔注3〕 OPコードの注記の意味

Ⓐ		Ⓑ		Ⓒ		Ⓓ		Ⓔ		Ⓕ		Ⓖ				Ⓗ	
Reg	ss dd	Reg	qq	Reg	pp	Reg	rr	Reg	r,r′	Bit	b	cc		Condition	Flag	P	t
BC	00	BC	00	BC	00	BC	00	B	000	0	000	000	NZ	Non Zero	Z	00H	000
DE	01	DE	01	DE	01	DE	01	C	001	1	001	001	Z	Zero	Z	08H	001
HL	10	HL	10	IX	10	IY	10	D	010	2	010	010	NC	Non Carry	C	10H	010
SP	11	AF	11	SP	11	SP	11	E	011	3	011	011	C	Carry	C	18H	011
								H	100	4	100	100	PO	Parity Odd	P/V	20H	100
								L	101	5	101	101	PE	Parity Even	P/V	28H	101
								A	111	6	110	110	P	Sign Positive	S	30H	110
										7	111	111	M	Sign Negative	S	38H	111

d, n：8ビット・イミーディエート・データ
e　：相対アドレシングの変位置
e−2：eの実効変位置

[1] インストラクション・セット

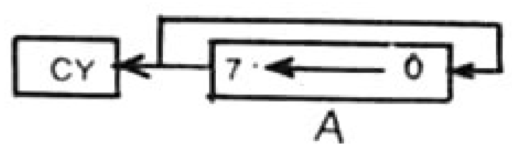
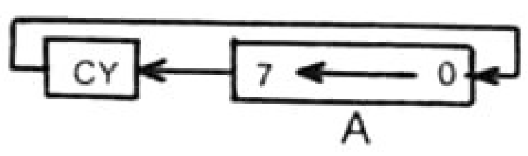
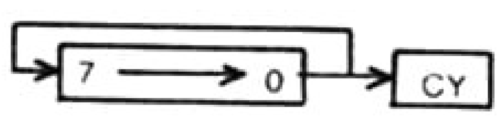
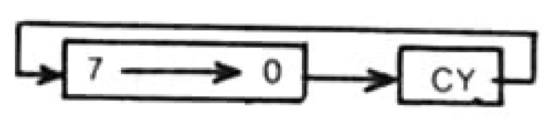
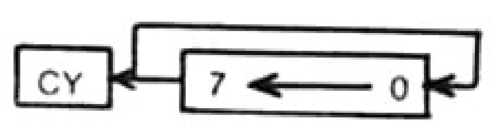
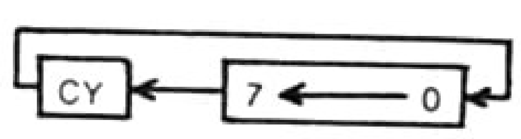
命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OPコード 76	543	210	
8ビット・ロード命令	LD r, r′	r←r′	•	•	•	•	•	•	1	4	01	r	r′	Ⓔ
	LD r, n	r←n	•	•	•	•	•	•	2	7	00	r	110	Ⓔ
											←	n	→	
	LD r, (HL)	r←(HL)	•	•	•	•	•	•	1	7	01	r	110	Ⓔ
	LD r, (IX+d)	r←(IX+d)	•	•	•	•	•	•	3	19	11	011	101	
											01	r	110	Ⓔ
											←	d	→	
	LD r, (IY+d)	r←(IY+d)	•	•	•	•	•	•	3	19	11	111	101	
											01	r	110	Ⓔ
											←	d	→	
	LD (HL), r	(HL)←r	•	•	•	•	•	•	1	7	01	110	r	Ⓔ
	LD (IX+d), r	(IX+d)←r	•	•	•	•	•	•	3	19	11	011	101	
											01	110	r	Ⓔ
											←	d	→	
	LD (IY+d), r	(IY+d)←r	•	•	•	•	•	•	3	19	11	111	101	
											01	110	r	Ⓔ
											←	d	→	
	LD (HL), n	(HL)←n	•	•	•	•	•	•	2	10	00	110	110	
											←	n	→	
	LD (IX+d), n	(IX+d)←n	•	•	•	•	•	•	4	19	11	011	101	
											00	110	110	
											←	d	→	
											←	n	→	
	LD (IY+d), n	(IY+d)←n	•	•	•	•	•	•	4	19	11	111	101	
											00	110	110	
											←	d	→	
											←	n	→	
	LD A,(BC)	A←(BC)	•	•	•	•	•	•	1	7	00	001	010	
	LD A,(DE)	A←(DE)	•	•	•	•	•	•	1	7	00	011	010	
	LD A,(nn)	A←(nn)	•	•	•	•	•	•	3	13	00	111	010	
											←	n	→	
											←	n	→	
	LD (BC), A	(BC)←A	•	•	•	•	•	•	1	7	00	000	010	
	LD (DE), A	(DE)←A	•	•	•	•	•	•	1	7	00	010	010	
	LD (nn), A	(nn)←A	•	•	•	•	•	•	3	13	00	110	010	
											←	n	→	
											←	n	→	
	LD A, I	A←I	↕	↕	0	IFF	0	•	2	9	11	101	101	
											01	010	111	
	LD A, R	A←R	↕	↕	0	IFF	0	•	2	9	11	101	101	
											01	011	111	
	LD I, A	I←A	•	•	•	•	•	•	2	9	11	101	101	
											01	000	111	
	LD R, A	R←A	•	•	•	•	•	•	2	9	11	101	101	
											01	001	111	
16ビット・ロード命令	LD dd, nn	dd←nn	•	•	•	•	•	•	3	10	00	dd0	001	Ⓐ
											←	n	→	
											←	n	→	
	LD IX, nn	IX←nn	•	•	•	•	•	•	4	14	11	011	101	
											00	100	001	
											←	n	→	
											←	n	→	

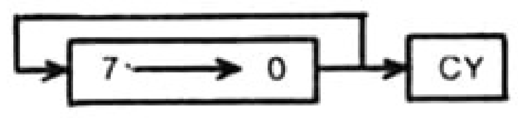
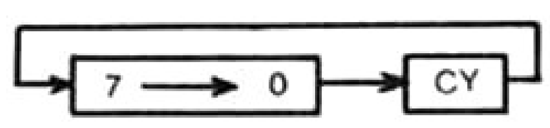
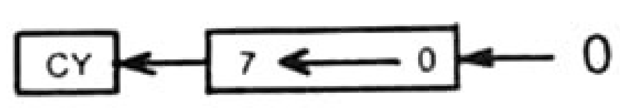
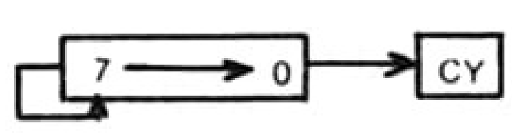
命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OPコード 76	543	210	
16ビット・ロード命令	LD IY, nn	IY←nn	•	•	•	•	•	•	4	14	11 00 ← ←	111 100 n n	101 001 → →	
	LD HL, (nn)	H←(nn+1) L←(nn)	•	•	•	•	•	•	3	16	00 ← ←	101 n n	010 → →	
	LD dd, (nn)	dd_H←(nn+1) dd_L←(nn)	•	•	•	•	•	•	4	20	11 01 ← ←	101 dd1 n n	101 011 → →	Ⓐ
	LD IX, (nn)	IX_H←(nn+1) IX_L←(nn)	•	•	•	•	•	•	4	20	11 00 ← ←	011 101 n n	101 010 → →	
	LD IY, (nn)	IY_H←(nn+1) IY_L←(nn)	•	•	•	•	•	•	4	20	11 00 ← ←	111 101 n n	101 010 → →	
	LD (nn), HL	(nn+1)←H (nn)←L	•	•	•	•	•	•	3	16	00 ← ←	100 n n	010 → →	
	LD (nn), dd	(nn+1)←dd_H (nn)←dd_L	•	•	•	•	•	•	4	20	11 01 ← ←	101 dd0 n n	101 011 → →	Ⓐ
	LD (nn), IX	(nn+1)←IX_H (nn)←IX_L	•	•	•	•	•	•	4	20	11 00 ← ←	011 100 n n	101 010 → →	
	LD (nn), IY	(nn+1)←IY_H (nn)←IY_L	•	•	•	•	•	•	4	20	11 00 ← ←	111 100 n n	101 010 → →	
	LD SP, HL	SP←HL	•	•	•	•	•	•	1	6	11	111	001	
	LD SP, IX	SP←IX	•	•	•	•	•	•	2	10	11 11	011 111	101 001	
	LD SP, IY	SP←IY	•	•	•	•	•	•	2	10	11 11	111 111	101 001	
	PUSH qq	(SP−2)←qq_L (SP−1)←qq_H	•	•	•	•	•	•	1	11	11	qq0	101	Ⓑ
	PUSH IX	(SP−2)←IX_L (SP−1)←IX_H	•	•	•	•	•	•	2	15	11 11	011 100	101 101	
	PUSH IY	(SP−2)←IY_L (SP−1)←IY_H	•	•	•	•	•	•	2	15	11 11	111 100	101 101	
	POP qq	qq_H←(SP+1) qq_L←(SP)	•	•	•	•	•	•	1	10	11	qq0	001	Ⓑ
	POP IX	IX_H←(SP+1) IX_L←(SP)	•	•	•	•	•	•	2	14	11 11	011 100	101 001	
	POP IY	IY_H←(SP+1) IY_L←(SP)	•	•	•	•	•	•	2	14	11 11	111 100	101 001	
エクスチェンジ命令	EX DE, HL	DE↔HL	•	•	•	•	•	•	1	4	11	101	011	
	EX AF, AF'	AF↔AF'	•	•	•	•	•	•	1	4	00	001	000	

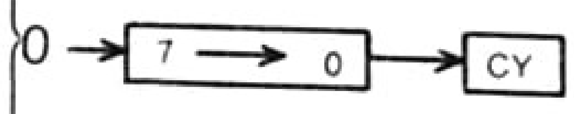
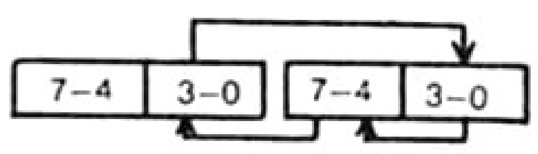
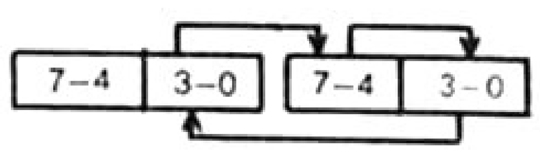
命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OP コード 76	543	210	
エクスチェンジ命令	EXX	(BC↔BC′ DE↔DE′ HL↔HL′)	•	•	•	•	•	•	1	4	11	011	001	
	EX (SP), HL	H↔(SP+1) L↔(SP)	•	•	•	•	•	•	1	19	11	100	011	
	EX (SP), IX	IX_H↔(SP+1) IX_L↔(SP)	•	•	•	•	•	•	2	23	11 11	011 100	101 011	
	EX (SP), IY	IY_H↔(SP+1) IY_L↔(SP)	•	•	•	•	•	•	2	23	11 11	111 100	101 011	
ブロック転送命令	LDI	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1	•	•	0	① ↕	0	•	2	16	11 10	101 100	101 000	
	LDIR	(DE)←(HL) DE←DE+1 HL←HL+1 BC←BC−1 until BC=0	•	•	0	0	0	•	2	21 if BC≠0 16 if BC=0	11 10	101 110	101 000	
	LDD	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1	•	•	0	① ↕	0	•	2	16	11 10	101 101	101 000	
	LDDR	(DE)←(HL) DE←DE−1 HL←HL−1 BC←BC−1 until BC=0	•	•	0	0	0	•	2	21 if BC≠0 16 if BC=0	11 10	101 111	101 000	
ブロック・サーチ命令	CPI	A−(HL) HL←HL+1 BC←BC−1	↕	② ↕	↕	① ↕	1	•	2	16	11 10	101 100	101 001	
	CPIR	A−(HL) HL←HL+1 BC←BC−1 until A=(HL) or BC=0	↕	② ↕	↕	① ↕	1	•	2	21 if BC≠0 and A≠(HL) 16 if BC=0 or A=(HL)	11 10	101 110	101 001	
	CPD	A−(HL) HL←HL−1 BC←BC−1	↕	② ↕	↕	① ↕	1	•	2	16	11 10	101 101	101 001	
	CPDR	A−(HL) HL←HL−1 BC←BC−1 until A=(HL) or BC=0	↕	② ↕	↕	① ↕	1	•	2	21 if BC≠0 and A≠(HL) 16 if BC=0 or A=(HL)	11 10	101 111	101 001	
8ビット算術論理演算命令	ADD A, r	A←A+r	↕	↕	↕	V	0	↕	1	4	10	000	r	Ⓔ
	ADD A, n	A←A+n	↕	↕	↕	V	0	↕	2	7	11 ←	000 n	110 →	
	ADD A, (HL)	A←A+(HL)	↕	↕	↕	V	0	↕	1	7	10	000	110	
	ADD A, (IX+d)	A←A+(IX+d)	↕	↕	↕	V	0	↕	3	19	11 10 ←	011 000 d	101 110 →	

命令群	ニモニック	オペレーション	フラグ						バイト	ステート	OPコード			
			S	Z	H	P/V	N	C			76	543	210	
8ビット算術論理演算命令	ADD, A, (IY+d)	A←A+(IY+d)	↕	↕	↕	V	0	↕	3	19	11	111	101	
											10	000	110	
											←	d	→	
	ADC A, r	A←A+r+CY	↕	↕	↕	V	0	↕	1	4	10	001	r	Ⓔ
	ADC A, n	A←A+n+CY	↕	↕	↕	V	0	↕	2	7	11	001	110	
											←	n	→	
	ADC A, (HL)	A←A+(HL)+CY	↕	↕	↕	V	0	↕	1	7	10	001	110	
	ADC A, (IX+d)	A←A+(IX+d)+CY	↕	↕	↕	V	0	↕	3	19	11	011	101	
											10	001	110	
											←	d	→	
	ADC A, (IY+d)	A←A+(IY+d)+CY	↕	↕	↕	V	0	↕	3	19	11	111	101	
											10	001	110	
											←	d	→	
	SUB r	A←A−r	↕	↕	↕	V	1	↕	1	4	10	010	r	Ⓔ
	SUB n	A←A−n	↕	↕	↕	V	1	↕	2	7	11	010	110	
											←	n	→	
	SUB (HL)	A←A−(HL)	↕	↕	↕	V	1	↕	1	7	10	010	110	
	SUB (IX+d)	A←A−(IX+d)	↕	↕	↕	V	1	↕	3	19	11	011	101	
											10	010	110	
											←	d	→	
	SUB (IY+d)	A←A−(IY+d)	↕	↕	↕	V	1	↕	3	19	11	111	101	
											10	010	110	
											←	d	→	
	SBC A, r	A←A−r−CY	↕	↕	↕	V	1	↕	1	4	10	011	r	Ⓔ
	SBC A, n	A←A−n−CY	↕	↕	↕	V	1	↕	2	7	11	011	110	
											←	n	→	
	SBC A, (HL)	A←A−(HL)−CY	↕	↕	↕	V	1	↕	1	7	10	011	110	
	SBC A, (IX+d)	A←A−(IX+d)−CY	↕	↕	↕	V	1	↕	3	19	11	011	101	
											10	011	110	
											←	d	→	
	SBC A, (IY+d)	A←A−(IY+d)−CY	↕	↕	↕	V	1	↕	3	19	11	111	101	
											10	011	110	
											←	d	→	
	AND r	A←A∧r	↕	↕	1	P	0	0	1	4	10	100	r	Ⓔ
	AND n	A←A∧n	↕	↕	1	P	0	0	2	7	11	100	110	
											←	n	→	
	AND (HL)	A←A∧(HL)	↕	↕	1	P	0	0	1	7	10	100	110	
	AND (IX+d)	A←A∧(IX+d)	↕	↕	1	P	0	0	3	19	11	011	101	
											10	100	110	
											←	d	→	
	AND (IY+d)	A←A∧(IY+d)	↕	↕	1	P	0	0	3	19	11	111	101	
											10	100	110	
											←	d	→	
	OR r	A←A∨r	↕	↕	0	P	0	0	1	4	10	110	r	Ⓔ
	OR n	A←A∨n	↕	↕	0	P	0	0	2	7	11	110	110	
											←	n	→	
	OR (HL)	A←A∨(HL)	↕	↕	0	P	0	0	1	7	10	110	110	
	OR (IX+d)	A←A∨(IX+d)	↕	↕	0	P	0	0	3	19	11	011	101	
											10	110	110	
											←	d	→	
	OR (IY+d)	A←A∨(IY+d)	↕	↕	0	P	0	0	3	19	11	111	101	
											10	110	110	
											←	d	→	
	XOR r	A←A⊻r	↕	↕	0	P	0	0	1	4	10	101	r	Ⓔ

命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OPコード 76	543	210	
8ビット算術論理演算命令	XOR n	A←A∀n	↕	↕	0	P	0	0	2	7	11	101	110	
											←	n	→	
	XOR (HL)	A←A∀(HL)	↕	↕	0	P	0	0	1	7	10	101	110	
	XOR (IX+d)	A←A∀(IX+d)	↕	↕	0	P	0	0	3	19	11	011	101	
											10	101	110	
											←	d	→	
	XOR (IY+d)	A←A∀(IY+d)	↕	↕	0	P	0	0	3	19	11	111	101	
											10	101	110	
											←	d	→	
	CP r	A−r	↕	↕	↕	V	1	↕	1	4	10	111	r	Ⓔ
	CP n	A−n	↕	↕	↕	V	1	↕	2	7	11	111	110	
											←	n	→	
	CP (HL)	A−(HL)	↕	↕	↕	V	1	↕	1	7	10	111	110	
	CP (IX+d)	A−(IX+d)	↕	↕	↕	V	1	↕	3	19	11	011	101	
											10	111	110	
											←	d	→	
	CP (IY+d)	A−(IY+d)	↕	↕	↕	V	1	↕	3	19	11	111	101	
											10	111	110	
											←	d	→	
	INC r	r←r+1	↕	↕	↕	V	0	•	1	4	00	r	100	Ⓔ
	INC (HL)	(HL)←(HL)+1	↕	↕	↕	V	0	•	1	11	00	110	100	
	INC (IX+d)	(IX+d)←(IX+d)+1	↕	↕	↕	V	0	•	3	23	11	011	101	
											00	110	100	
											←	d	→	
	INC (IY+d)	(IY+d)←(IX+d)+1	↕	↕	↕	V	0	•	3	23	11	111	101	
											00	110	100	
											←	d	→	
	DEC r	r←r−1	↕	↕	↕	V	1	•	1	4	00	r	101	Ⓔ
	DEC (HL)	(HL)←(HL)−1	↕	↕	↕	V	1	•	1	11	00	110	101	
	DEC (IX+d)	(IX+d)←(IX+d)−1	↕	↕	↕	V	1	•	3	23	11	011	101	
											00	110	101	
											←	d	→	
	DEC (IY+d)	(IY+d)←(IY+d)−1	↕	↕	↕	V	1	•	3	23	11	111	101	
											00	110	101	
											←	d	→	
16ビット算術演算命令	ADD HL, ss	HL←HL+ss	•	•	×	•	0	↕	1	11	00	ss1	001	Ⓐ
	ADC HL, ss	HL←HL+ss+CY	↕	↕	×	V	0	↕	2	15	11	101	101	
											01	ss1	010	Ⓐ
	SBC HL, ss	HL←HL−ss−CY	↕	↕	×	V	1	↕	2	15	11	101	101	
											01	ss0	010	Ⓐ
	ADD IX, pp	IX←IX+pp	•	•	×	•	0	↕	2	15	11	011	101	
											00	pp1	001	Ⓒ
	ADD IY, rr	IY←IY+rr	•	•	×	•	0	↕	2	15	11	111	101	
											00	rr1	001	Ⓓ
	INC ss	ss←ss+1	•	•	•	•	•	•	1	6	00	ss0	011	Ⓐ
	INC IX	IX←IX+1	•	•	•	•	•	•	2	10	11	011	101	
											00	100	011	
	INC IY	IY←IY+1	•	•	•	•	•	•	2	10	11	111	101	
											00	100	011	
	DEC ss	ss←ss−1	•	•	•	•	•	•	1	6	00	ss1	011	Ⓐ
	DEC IX	IX←IX−1	•	•	•	•	•	•	2	10	11	011	101	
											00	101	011	
	DEC IY	IY←IY−1	•	•	•	•	•	•	2	10	11	111	101	
											00	101	011	

命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OP コード 76 543 210	
アキュムレータ操作命令	DAA	Decimal Adjust Acc	↕	↕	↕	P	•	↕	1	4	00 100 111	
	CPL	A←$\overline{A}$	•	•	1	•	1	•	1	4	00 101 111	
	NEG	A←$\overline{A}$+1	↕	↕	↕	V	1	↕	2	8	11 101 101 01 000 100	
	CCF	CY←$\overline{CY}$	•	•	×	•	0	↕	1	4	00 111 111	
	SCF	CY←1	•	•	0	•	0	1	1	4	00 110 111	
CPUコントロール命令	NOP	No Operation	•	•	•	•	•	•	1	4	00 000 000	
	HALT	CPU Halted	•	•	•	•	•	•	1	4	01 110 110	
	DI	IFF←0	•	•	•	•	•	•	1	4	11 110 011	
	EI	IFF←1	•	•	•	•	•	•	1	4	11 111 011	
	IM 0	Set Interrupt Mode 0	•	•	•	•	•	•	2	8	11 101 101 01 000 110	
	IM 1	Set Interrupt Mode 1	•	•	•	•	•	•	2	8	11 101 101 01 010 110	
	IM 2	Set Interrupt Mode 2	•	•	•	•	•	•	2	8	11 101 101 01 011 110	
ローテート・シフト命令	RLCA		•	•	0	•	0	↕	1	4	00 000 111	
	RLA		•	•	0	•	0	↕	1	4	00 010 111	
	RRCA		•	•	0	•	0	↕	1	4	00 001 111	
	RRA		•	•	0	•	0	↕	1	4	00 011 111	
	RLC r		↕	↕	0	P	0	↕	2	8	11 001 011 00 000 r	Ⓔ
	RLC (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 000 110	
	RLC (IX+d)	 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 000 110	
	RLC (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 000 110	
	RL r		↕	↕	0	P	0	↕	2	8	11 001 011 00 010 r	
	RL (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 010 110	Ⓔ
	RL (IX+d)	 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 010 110	

命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OPコード 76 543 210	
ローテート・シフト命令	RL (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 010 110	
	RRC r		↕	↕	0	P	0	↕	2	8	11 001 011 00 001 r	Ⓔ
	RRC (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 001 110	
	RRC (IX+d)	 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 001 110	
	RRC (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 001 110	
	RR r		↕	↕	0	P	0	↕	2	8	11 001 011 00 011 r	Ⓔ
	RR (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 011 110	
	RR (IX+d)	 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 011 110	
	RR (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 011 110	
	SLA r		↕	↕	0	P	0	↕	2	8	11 001 011 00 100 r	Ⓔ
	SLA (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 100 110	
	SLA (IX+d)	 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 100 110	
	SLA (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 100 110	
	SRA r		↕	↕	0	P	0	↕	2	8	11 001 011 00 101 r	Ⓔ
	SRA (HL)		↕	↕	0	P	0	↕	2	15	11 001 011 00 101 110	
	SRA (IX+d)	 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	4	23	11 011 101 11 001 011 ← d → 00 101 110	
	SRA (IY+d)		↕	↕	0	P	0	↕	4	23	11 111 101 11 001 011 ← d → 00 101 110	

命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OP コード 76	543	210	
ローテート・シフト命令	SRL r	 r, (HL), (IX+d), (IY+d)	↕	↕	0	P	0	↕	2	8	11 00	001 111	011 r	Ⓔ
	SRL (HL)		↕	↕	0	P	0	↕	2	15	11 00	001 111	011 110	
	SRL (IX+d)		↕	↕	0	P	0	↕	4	23	11 11 ← 00	011 001 d 111	101 011 → 110	
	SRL (IY+d)		↕	↕	0	P	0	↕	4	23	11 11 ← 00	111 001 d 111	101 011 → 110	
	RLD	A  (HL)	↕	↕	0	P	0	•	2	18	11 01	101 101	101 111	
	RRD	A  (HL)	↕	↕	0	P	0	•	2	18	11 01	101 100	101 111	
ビット操作命令	BIT b, r	$Z \leftarrow \overline{r_b}$	×	↕	1	×	0	•	2	8	11 01	001 b	011 r	ⒺⒻ
	BIT b, (HL)	$Z \leftarrow \overline{(HL)}_b$	×	↕	1	×	0	•	2	12	11 01	001 b	011 110	Ⓕ
	BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)}_b$	×	↕	1	×	0	•	4	20	11 11 ← 01	011 001 d b	101 011 → 110	Ⓕ
	BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)}_b$	×	↕	1	×	0	•	4	20	11 11 ← 01	111 001 d b	101 011 → 110	Ⓕ
	SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	2	8	11 11	001 b	011 r	ⒺⒻ
	SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	2	15	11 11	001 b	011 110	Ⓕ
	SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	•	•	•	•	4	23	11 11 ← 11	011 001 d b	101 011 → 110	Ⓕ
	SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	•	•	•	•	4	23	11 11 ← 11	111 001 d b	101 011 → 110	Ⓕ
	RES b, r	$r_b \leftarrow 0$	•	•	•	•	•	•	2	8	11 10	001 b	011 r	ⒺⒻ
	RES b, (HL)	$(HL)_b \leftarrow 0$	•	•	•	•	•	•	2	15	11 10	001 b	011 110	Ⓕ
	RES b, (IX+d)	$(IX+d)_b \leftarrow 0$	•	•	•	•	•	•	4	23	11 11 ← 10	011 001 d b	101 011 → 110	Ⓕ
	RES b, (IY+d)	$(IY+d)_b \leftarrow 0$	•	•	•	•	•	•	4	23	11 11 ← 10	111 001 d b	101 011 → 110	Ⓕ

命令群	ニモニック	オペレーション	S	Z	H	P/V	N	C	バイト	ステート	76	543	20	
ジャンプ・コール・リターン命令	JP nn	PC←nn	•	•	•	•	•	•	3	10	11 ← ←	000 n n	011 → →	
	JP cc, nn	If cc is true PC←nn otherwise continue	•	•	•	•	•	•	3	10	11 ← ←	cc n n	010 → →	Ⓖ
	JR e	PC←PC+e	•	•	•	•	•	•	2	12	00 ←	011 e−2	000 →	
	JR C, e	If C=0 continue If C=1 PC←PC+e	•	•	•	•	•	•	2	7 if C=0 12 if C=1	00 ←	111 e−2	000 →	
	JR NC, e	If C=1 continue If C=0 PC←PC+e	•	•	•	•	•	•	2	7 if C=1 12 if C=0	00 ←	110 e−2	000 →	
	JR Z, e	If Z=0 continue If Z=1 PC←PC+e	•	•	•	•	•	•	2	7 if Z=0 12 if Z=1	00 ←	101 e−2	000 →	
	JR NZ, e	If Z=1 continue If Z=0 PC←PC+e	•	•	•	•	•	•	2	7 if Z=1 12 if Z=0	00 ←	100 e−2	000 →	
	JP (HL)	PC←HL	•	•	•	•	•	•	1	4	11	101	001	
	JP (IX)	PC←IX	•	•	•	•	•	•	2	8	11 11	011 101	101 001	
	JP (IY)	PC←IY	•	•	•	•	•	•	2	8	11 11	111 101	101 001	
	DJNZ e	B←B−1 if B=0 continue if B≠0 PC←PC+e	•	•	•	•	•	•	2	8 if B=0 13 if B≠0	00 ←	010 e−2	000 →	
	CALL nn	(SP−1)←PC_H (SP−2)←PC_L PC←nn	•	•	•	•	•	•	3	17	11 ← ←	001 n n	101 → →	
	CALL cc, nn	If cc is false continue otherwise same as CALL nn	•	•	•	•	•	•	3	10 if cc is false 17 if cc is true	11 ← ←	cc n n	100 → →	Ⓖ
	RET	PC_L←(SP) PC_H←(SP+1)	•	•	•	•	•	•	1	10	11	001	001	
	RET cc	If cc is false continue otherwise same as RET	•	•	•	•	•	•	1	5 if cc is false 11 if cc is true	11	cc	000	Ⓖ
	RETI	Return from interrupt	•	•	•	•	•	•	2	14	11 01	101 001	101 101	
	RETN	Return from non maskable interrupt	•	•	•	•	•	•	2	14	11 01	101 000	101 101	
	RST p	(SP−1)←PC_H (SP−2)←PC_L PC_H←0 PC_L←p	•	•	•	•	•	•	1	11	11	t	111	Ⓗ
入出力命令	IN A, n	A←(n) A_{0-7}←n A_{8-15}←A	•	•	•	•	•	•	2	11	11 ←	011 n	011 →	
	IN r, (C)	r←(C) if r=110 only the flags will be affected A_{0-7}←C A_{8-15}←B	↕	↕	↕	P	0	•	2	12	11 01	101 r	101 000	Ⓔ
	INI	(HL)←(C) B←B−1 HL←HL+1 A_{0-7}←C A_{8-15}←B	×	③ ↕	×	×	1	•	2	16	11 10	101 100	101 010	

命令群	ニモニック	オペレーション	フラグ S	Z	H	P/V	N	C	バイト	ステート	OPコード 76	543	210	
入出力命令	INIR	(HL)←(C) B←B−1 HL←HL+1 until B=0 A_{0-7}←C A_{8-15}←B	×	1	×	×	1	•	2	21 if B≠0 16 if B=0	11 10	101 110	101 010	
	IND	(HL)←(C) B←B−1 HL←HL−1 A_{0-7}←C A_{8-15}←B	×	③ ↕	×	×	1	•	2	16	11 10	101 101	101 010	
	INDR	(HL)←(C) B←B−1 HL←HL−1 until B=0 A_{0-7}←C A_{8-15}←B	×	1	×	×	1	•	2	21 if B≠0 16 if B=0	11 10	101 111	101 010	
	OUT n, A	(n)←A A_{0-7}←n A_{8-15}←A	•	•	•	•	•	•	2	11	11 ←	010 n	011 →	
	OUT (C), r	(C)←r A_{0-7}←C A_{8-15}←B	•	•	•	•	•	•	2	12	11 01	101 r	101 001	Ⓔ
	OUTI	(C)←(HL) B←B−1 HL←HL+1 A_{0-7}←C A_{8-15}←B	×	③ ↕	×	×	1	•	2	16	11 10	101 100	101 011	
	OTIR	(C)←(HL) B←B−1 HL←HL+1 until B=0 A_{0-7}←C A_{8-15}←B	×	1	×	×	1	•	2	21 if B≠0 16 if B=0	11 10	101 110	101 011	
	OUTD	(C)←(HL) B←B−1 HL←HL−1 A_{0-7}←C A_{8-15}←B	×	③ ↕	×	×	1	•	2	16	11 10	101 101	101 011	
	OTDR	(C)←(HL) B←B−1 HL←HL−1 until B=0 A_{0-7}←C A_{8-15}←B	×	1	×	×	1	•	2	21 if B≠0 16 if B=0	11 10	101 111	101 011	

〔2〕 マシン語↔ニモニック対応表

00	NOP		40	LD	B, B	80	ADD	A, B	C0	RET	NZ
01	LD	BC, nn	41	LD	B, C	81	ADD	A, C	C1	POP	BC
02	LD	(BC), A	42	LD	B, D	82	ADD	A, D	C2	JP	NZ, nn
03	INC	BC	43	LD	B, E	83	ADD	A, E	C3	JP	nn
04	INC	B	44	LD	B, H	84	ADD	A, H	C4	CALL	NZ, nn
05	DEC	B	45	LD	B, L	85	ADD	A, L	C5	PUSH	BC
06	LD	B, n	46	LD	B, (HL)	86	ADD	A, (HL)	C6	ADD	A, n
07	RLCA		47	LD	B, A	87	ADD	A, A	C7	RST	00H
08	EX	AF, AF	48	LD	C, B	88	ADC	A, B	C8	RET	Z
09	ADD	HL, BC	49	LD	C, C	89	ADC	A, C	C9	RET	
0A	LD	A, (BC)	4A	LD	C, D	8A	ADC	A, D	CA	JP	Z, nn
0B	DEC	BC	4B	LD	C, E	8B	ADC	A, E	CB		
0C	INC	C	4C	LD	C, H	8C	ADC	A, H	CC	CALL	Z, nn
0D	DEC	C	4D	LD	C, L	8D	ADC	A, L	CD	CALL	nn
0E	LD	C, n	4E	LD	C, (HL)	8E	ADC	A, (HL)	CE	ADC	A, n
0F	RRCA		4F	LD	C, A	8F	ADC	A, A	CF	RST	08H
10	DJNZ	e	50	LD	D, B	90	SUB	B	D0	RET	NC
11	LD	DE, nn	51	LD	D, C	91	SUB	C	D1	POP	DE
12	LD	(DE), A	52	LD	D, D	92	SUB	D	D2	JP	NC, nn
13	INC	DE	53	LD	D, E	93	SUB	E	D3	OUT	n, A
14	INC	D	54	LD	D, H	94	SUB	H	D4	CALL	NC, nn
15	DEC	D	55	LD	D, L	95	SUB	L	D5	PUSH	DE
16	LD	D, n	56	LD	D, (HL)	96	SUB	(HL)	D6	SUB	n
17	RLA		57	LD	D, A	97	SUB	A	D7	RST	10H
18	JR	e	58	LD	E, B	98	SBC	A, B	D8	RET	C
19	ADD	HL, DE	59	LD	E, C	99	SBC	A, C	D9	EXX	
1A	LD	A, (DE)	5A	LD	E, D	9A	SBC	A, D	DA	JP	C, nn
1B	DEC	DE	5B	LD	E, E	9B	SBC	A, E	DB	IN	A, n
1C	INC	E	5C	LD	E, H	9C	SBC	A, H	DC	CALL	C, nn
1D	DEC	E	5D	LD	E, L	9D	SBC	A, L	DD		
1E	LD	E, n	5E	LD	E, (HL)	9E	SBC	A, (HL)	DE	SBC	A, n
1F	RRA		5F	LD	E, A	9F	SBC	A, A	DF	RST	18H
20	JR	NZ, e	60	LD	H, B	A0	AND	B	E0	RET	PO
21	LD	HL, nn	61	LD	H, C	A1	AND	C	E1	POP	HL
22	LD	(nn), HL	62	LD	H, D	A2	AND	D	E2	JP	PO, nn
23	INC	HL	63	LD	H, E	A3	AND	E	E3	EX	(SP), HL
24	INC	H	64	LD	H, H	A4	AND	H	E4	CALL	PO, nn
25	DEC	H	65	LD	H, L	A5	AND	L	E5	PUSH	HL
26	LD	H, n	66	LD	H, (HL)	A6	AND	(HL)	E6	AUD	n
27	DAA		67	LD	H, A	A7	AND	A	E7	RST	20H
28	JR	Z, e	68	LD	L, B	A8	XOR	B	E8	RET	PE
29	ADD	HL, HL	69	LD	L, C	A9	XOR	C	E9	JP	(HL)
2A	LD	HL, (nn)	6A	LD	L, D	AA	XOR	D	EA	JP	PE, nn
2B	DEC	HL	6B	LD	L, E	AB	XOR	E	EB	EX	DE, HL
2C	INC	L	6C	LD	L, H	AC	XOR	H	EC	CALL	PE, nn
2D	DEC	L	6D	LD	L, L	AD	XOR	L	ED		
2E	LD	L, n	6E	LD	L, (HL)	AE	XOR	(HL)	EE	XOR	n
2F	CPL		6F	LD	L, A	AF	XOR	A	EF	RST	28H
30	JR	NC, e	70	LD	(HL), B	B0	OR	B	F0	RET	P
31	LD	SP, nn	71	LD	(HL), C	B1	OR	C	F1	POP	AF
32	LD	(nn), A	72	LD	(HL), D	B2	OR	D	F2	JP	P, nn
33	INC	SP	73	LD	(HL), E	B3	OR	E	F3	DI	
34	INC	(HL)	74	LD	(HL), H	B4	OR	H	F4	CALL	P, nn
35	DEC	(HL)	75	LD	(HL), L	B5	OR	L	F5	PUSH	AF
36	LD	(HL), n	76	HALT		B6	OR	(HL)	F6	OR	n
37	SCF		77	LD	(HL), A	B7	OR	A	F7	RST	30H
38	JR	C, e	78	LD	A, B	B8	CP	B	F8	RET	M
39	ADD	HL, SP	79	LD	A, C	B9	CP	C	F9	LD	SP, HL
3A	LD	A, (nn)	7A	LD	A, D	BA	CP	D	FA	JP	M, nn
3B	DEC	SP	7B	LD	A, E	BB	CP	E	FB	EI	
3C	INC	A	7C	LD	A, H	BC	CP	H	FC	CALL	M, nn
3D	DEC	A	7D	LD	A, L	BD	CP	L	FD		
3E	LD	A, n	7E	LD	A, (HL)	BE	CP	(HL)	FE	CP	n
3F	CCF		7F	LD	A, A	BF	CP	A	FF	RST	38H

CB ××											
00	RLC	B	40	BIT	0, B	80	RES	0, B	C0	SET	0, B
01	RLC	C	41	BIT	0, C	81	RES	0, C	C1	SET	0, C
02	RLC	D	42	BIT	0, D	82	RES	0, D	C2	SET	0, D
03	RLC	E	43	BIT	0, E	83	RES	0, E	C3	SET	0, E
04	RLC	H	44	BIT	0, H	84	RES	0, H	C4	SET	0, H
05	RLC	L	45	BIT	0, L	85	RES	0, L	C5	SET	0, L
06	RLC	(HL)	46	BIT	0, (HL)	86	RES	0, (HL)	C6	SET	0, (HL)
07	RLC	A	47	BIT	0, A	87	RES	0, A	C7	SET	0, A
08	RRC	B	48	BIT	1, B	88	RES	1, B	C8	SET	1, B
09	RRC	C	49	BIT	1, C	89	RES	1, C	C9	SET	1, C
0A	RRC	D	4A	BIT	1, D	8A	RES	1, D	CA	SET	1, D
0B	RRC	E	4B	BIT	1, E	8B	RES	1, E	CB	SET	1, E
0C	RRC	H	4C	BIT	1, H	8C	RES	1, H	CC	SET	1, H
0D	RRC	L	4D	BIT	1, L	8D	RES	1, L	CD	SET	1, L
0E	RRC	(HL)	4E	BIT	1, (HL)	8E	RES	1, (HL)	CE	SET	1, (HL)
0F	RRC	A	4F	BIT	1, A	8F	RES	1, A	CF	SET	1, A
10	RL	B	50	BIT	2, B	90	RES	2, B	D0	SET	2, B
11	RL	C	51	BIT	2, C	91	RES	2, C	D1	SET	2, C
12	RL	D	52	BIT	2, D	92	RES	2, D	D2	SET	2, D
13	RL	E	53	BIT	2, E	93	RES	2, E	D3	SET	2, E
14	RL	H	54	BIT	2, H	94	RES	2, H	D4	SET	2, H
15	RL	L	55	BIT	2, L	95	RES	2, L	D5	SET	2, L
16	RL	(HL)	56	BIT	2, (HL)	96	RES	2, (HL)	D6	SET	2, (HL)
17	RL	A	57	BIT	2, A	97	RES	2, A	D7	SET	2, A
18	RR	B	58	BIT	3, B	98	RES	3, B	D8	SET	3, B
19	RR	C	59	BIT	3, C	99	RES	3, C	D9	SET	3, C
1A	RR	D	5A	BIT	3, D	9A	RES	3, D	DA	SET	3, D
1B	RR	E	5B	BIT	3, E	9B	RES	3, E	DB	SET	3, E
1C	RR	H	5C	BIT	3, H	9C	RES	3, H	DC	SET	3, H
1D	RR	L	5D	BIT	3, L	9D	RES	3, L	DD	SET	3, L
1E	RR	(HL)	5E	BIT	3, (HL)	9E	RES	3, (HL)	DE	SET	3, (HL)
1F	RR	A	5F	BIT	3, A	9F	RES	3, A	DF	SET	3, A
20	SLA	B	60	BIT	4, B	A0	RES	4, B	E0	SET	4, B
21	SLA	C	61	BIT	4, C	A1	RES	4, C	E1	SET	4, C
22	SLA	D	62	BIT	4, D	A2	RES	4, D	E2	SET	4, D
23	SLA	E	63	BIT	4, E	A3	RES	4, E	E3	SET	4, E
24	SLA	H	64	BIT	4, H	A4	RES	4, H	E4	SET	4, H
25	SLA	L	65	BIT	4, L	A5	RES	4, L	E5	SET	4, L
26	SLA	(HL)	66	BIT	4, (HL)	A6	RES	4, (HL)	E6	SET	4, (HL)
27	SLA	A	67	BIT	4, A	A7	RES	4, A	E7	SET	4, A
28	SRA	B	68	BIT	5, B	A8	RES	5, B	E8	SET	5, B
29	SRA	C	69	BIT	5, C	A9	RES	5, C	E9	SET	5, C
2A	SRA	D	6A	BIT	5, D	AA	RES	5, D	EA	SET	5, D
2B	SRA	E	6B	BIT	5, E	AB	RES	5, E	EB	SET	5, E
2C	SRA	H	6C	BIT	5, H	AC	RES	5, H	EC	SET	5, H
2D	SRA	L	6D	BIT	5, L	AD	RES	5, L	ED	SET	5, L
2E	SRA	(HL)	6E	BIT	5, (HL)	AE	RES	5, (HL)	EE	SET	5, (HL)
2F	SRA	A	6F	BIT	5, A	AF	RES	5, A	EF	SET	5, A
30			70	BIT	6, B	B0	RES	6, B	F0	SET	6, B
31			71	BIT	6, C	B1	RES	6, C	F1	SET	6, C
32			72	BIT	6, D	B2	RES	6, D	F2	SET	6, D
33			73	BIT	6, E	B3	RES	6, E	F3	SET	6, E
34			74	BIT	6, H	B4	RES	6, H	F4	SET	6, H
35			75	BIT	6, L	B5	RES	6, L	F5	SET	6, L
36			76	BIT	6, (HL)	B6	RES	6, (HL)	F6	SET	6, (HL)
37			77	BIT	6, A	B7	RES	6, A	F7	SET	6, A
38	SRL	B	78	BIT	7, B	B8	RES	7, B	F8	SET	7, B
39	SRL	C	79	BIT	7, C	B9	RES	7, C	F9	SET	7, C
3A	SRL	D	7A	BIT	7, D	BA	RES	7, D	FA	SET	7, D
3B	SRL	E	7B	BIT	7, E	BB	RES	7, E	FB	SET	7, E
3C	SRL	H	7C	BIT	7, H	BC	RES	7, H	FC	SET	7, H
3D	SRL	L	7D	BIT	7, L	BD	RES	7, L	FD	SET	7, L
3E	SRL	(HL)	7E	BIT	7, (HL)	BE	RES	7, (HL)	FE	SET	7, (HL)
3F	SRL	A	7F	BIT	7, A	BF	RES	7, A	FF	SET	7, A

DD ××				
09			ADD	IX, BC
19			ADD	IX, DE
21			LD	IX, nn
22			LD	(nn), IX
23			INC	IX
29			ADD	IX, IX
2A			LD	IX, (nn)
2B			DEC	IX
34			INC	(IX+d)
35			DEC	(IX+d)
36			LD	(IX+d), n
39			ADD	IX, SP
46			LD	B, (IX+d)
4E			LD	C, (IX+d)
56			LD	D, (IX+d)
5E			LD	E, (IX+d)
66			LD	H, (IX+d)
6E			LD	L, (IX+d)
70			LD	(IX+d), B
71			LD	(IX+d), C
72			LD	(IX+d), D
73			LD	(IX+d), E
74			LD	(IX+d), H
75			LD	(IX+d), L
77			LD	(IX+d), A
7E			LD	A, (IX+d)
86			ADD	A, (IX+d)
8E			ADC	A, (IX+d)
96			SUB	(IX+d)
9E			SBC	A, (IX+d)
A6			AND	(IX+d)
AE			XOR	(IX+d)
B6			OR	(IX+d)
BE			CP	(IX+d)
CB	d	06	RLC	(IX+d)
CB	d	0E	RRC	(IX+d)
CB	d	16	RL	(IX+d)
CB	d	1E	RR	(IX+d)
CB	d	26	SLA	(IX+d)
CB	d	2E	SRA	(IX+d)
CB	d	3E	SRL	(IX+d)
CB	d	46	BIT	0, (IX+d)
CB	d	4E	BIT	1, (IX+d)
CB	d	56	BIT	2, (IX+d)
CB	d	5E	BIT	3, (IX+d)
CB	d	66	BIT	4, (IX+d)
CB	d	6E	BIT	5, (IX+d)
CB	d	76	BIT	6, (IX+d)
CB	d	7E	BIT	7, (IX+d)
CB	d	86	RES	0, (IX+d)
CB	d	8E	RES	1, (IX+d)
CB	d	96	RES	2, (IX+d)
CB	d	9E	RES	3, (IX+d)
CB	d	A6	RES	4, (IX+d)
CB	d	AE	RES	5, (IX+d)
CB	d	B6	RES	6, (IX+d)
CB	d	BE	RES	7, (IX+d)
CB	d	C6	SET	0, (IX+d)
CB	d	CE	SET	1, (IX+d)
CB	d	D6	SET	2, (IX+d)
CB	d	DE	SET	3, (IX+d)
CB	d	E6	SET	4, (IX+d)
CB	d	EE	SET	5, (IX+d)
CB	d	F6	SET	6, (IX+d)
CB	d	FE	SET	7, (IX+d)
E1			POP	IX
E3			EX	(SP), IX
E5			PUSH	IX
E9			JP	(IX)
F9			LD	SP, IX

ED ××		
40	IN	B, (C)
41	OUT	(C), B
42	SBC	HL, BC
43	LD	(nn), BC
44	NEG	
45	RETN	
46	IM	0
47	LD	I, A
48	IN	C, (C)
49	OUT	(C), C
4A	ADC	HL, BC
4B	LD	BC, (nn)
4D	RETI	
4F	LD	R, A
50	IN	D, (C)
51	OUT	(C), D
52	SBC	HL, DE
53	LD	(nn), DE
56	IM	1
57	LD	A, I
58	IN	E, (C)
59	OUT	(C), E
5A	ADC	HL, DE
5B	LD	DE, (nn)
5E	IM	2
5F	LD	A, R
60	IN	H, (C)
61	OUT	(C), H
62	SBC	HL, HL
67	RRD	
68	IN	L, (C)
69	OUT	(C), L
6A	ADC	HL, HL
6F	RLD	
72	SBC	HL, SP
73	LD	(nn), SP
78	IN	A, (C)
79	OUT	(C), A
7A	ADC	HL, SP
7B	LD	SP, (nn)
A0	LDI	
A1	CPI	
A2	INI	
A3	OUTI	
A8	LDD	
A9	CPD	
AA	IND	
AB	OUTD	
B0	LDIR	
B1	CPIR	
B2	INIR	
B3	OTIR	
B8	LDDR	
B9	CPDR	
BA	INDR	
BB	OTDR	

FD ××				
09			ADD	IY, BC
19			ADD	IY, DE
21			LD	IY, nn
22			LD	(nn), IY
23			INC	IY
29			ADD	IY, IY
2A			LD	IY, (nn)
2B			DEC	IY
34			INC	(IY+d)
35			DEC	(IY+d)
36			LD	(IY+d), n
39			ADD	IY, SP
46			LD	B, (IY+d)
4E			LD	C, (IY+d)
56			LD	D, (IY+d)
5E			LD	E, (IY+d)
66			LD	H, (IY+d)
6E			LD	L, (IY+d)
70			LD	(IY+d), B
71			LD	(IY+d), C
72			LD	(IY+d), D
73			LD	(IY+d), E
74			LD	(IY+d), H
75			LD	(IY+d), L
77			LD	(IY+d), A
7E			LD	A, (IY+d)
86			ADD	A, (IY+d)
8E			ADC	A, (IY+d)
96			SUB	(IY+d)
9E			SBC	A, (IY+d)
A6			AND	(IY+d)
AE			XOR	(IY+d)
B6			OR	(IY+d)
BE			CP	(IY+d)
CB	d	06	RLC	(IY+d)
CB	d	0E	RRC	(IY+d)
CB	d	16	RL	(IY+d)
CB	d	1E	RR	(IY+d)
CB	d	26	SLA	(IY+d)
CB	d	2E	SRA	(IY+d)
CB	d	3E	SRL	(IY+d)
CB	d	46	BIT	0, (IY+d)
CB	d	4E	BIT	1, (IY+d)
CB	d	56	BIT	2, (IY+d)
CB	d	5E	BIT	3, (IY+d)
CB	d	66	BIT	4, (IY+d)
CB	d	6E	BIT	5, (IY+d)
CB	d	76	BIT	6, (IY+d)
CB	d	7E	BIT	7, (IY+d)
CB	d	86	RES	0, (IY+d)
CB	d	8E	RES	1, (IY+d)
CB	d	96	RES	2, (IY+d)
CB	d	9E	RES	3, (IY+d)
CB	d	A6	RES	4, (IY+d)
CB	d	AE	RES	5, (IY+d)
CB	d	B6	RES	6, (IY+d)
CB	d	BE	RES	7, (IY+d)
CB	d	C6	SET	0, (IY+d)
CB	d	CE	SET	1, (IY+d)
CB	d	D6	SET	2, (IY+d)
CB	d	DE	SET	3, (IY+d)
CB	d	E6	SET	4, (IY+d)
CB	d	EE	SET	5, (IY+d)
CB	d	F6	SET	6, (IY+d)
CB	d	FE	SET	7, (IY+d)
E1			POP	IY
E3			EX	(SP), IY
E5			PUSH	IY
E9			JP	(IY)
F9			LD	SP, IY

〔3〕 ニモニック↔マシン語対照表

8ビット・ロード

＼ ×	I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX+d)	(IY+d)	(nn)	n
LD A, ×	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
LD B, ×			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
LD C, ×			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
LD D, ×			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
LD E, ×			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
LD H, ×			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
LD L, ×			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
LD (HL), ×			77	70	71	72	73	74	75							36 n
LD (BC), ×			02													
LD (DE), ×			12													
LD (IX+d), ×			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
LD (IY+d), ×			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
LD (nn), ×			32 n n													
LD I, ×			ED 47													
LD R, ×			ED 4F													

16ビット・ロード

＼ ×	AF	BC	DE	HL	SP	IX	IY	nn	(nn)
LD AF, ×									
LD BC, ×								01 n n	ED 4B n n
LD DE, ×								11 n n	ED 5B n n
LD HL, ×								21 n n	2A n n
LD SP, ×				F9		DD F9	FD F9	31 n n	ED 7B n n
LD IX, ×								DD 21 n n	DD 2A n n
LD IY, ×								FD 21 n n	FD 2A n n
LD (nn), ×		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n		
PUSH ×	F5	C5	D5	E5		DD E5	FD E5		
POP ×	F1	C1	D1	E1		DD E1	FD E1		

ブロック・転送

LDI	ED A0
LDIR	ED B0
LDD	ED A8
LDDR	ED B8

ブロック・サーチ

CPI	ED A1
CPIR	ED B1
CPD	ED A9
CPDR	ED B9

8ビット算術論理演算

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
ADD A, ×	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADC A, ×	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUB ×	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SBC A, ×	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
AND ×	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
XOR ×	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
OR ×	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
CP ×	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INC ×	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DEC ×	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

CPUコントロール

NOP	00
HALT	76
DI	F3
EI	FB
IM 0	ED 46
IM 1	ED 56
IM 2	ED 5E

16ビット算術演算

×	BC	DE	HL	SP	IX	IY
ADD HL, ×	09	19	29	39		
ADD IX, ×	DD 09	DD 19		DD 39	DD 29	
ADD IY, ×	FD 09	FD 19		FD 39		FD 29
ADC HL, ×	ED 4A	ED 5A	ED 6A	ED 7A		
SBC HL, ×	ED 42	ED 52	ED 62	ED 72		
INC ×	03	13	23	33	DD 23	FD 23
DEC ×	0B	1B	2B	3B	DD 2B	FD 2B

エクスチェンジ

EX AF, AF'	08
EX DE, HL	EB
EX (SP), HL	E3
EX (SP), IX	DD E3
EX (SP), IY	FD E3
EXX	D9

アキュムレータ操作

DAA	27
CPL	2F
NEG	ED 44
CCF	3F
SCF	37

ローテート，シフト

×	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
RLC ×	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
RRC ×	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
RL ×	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
RR ×	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
SLA ×	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
SRA ×	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
SRL ×	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
RLD								ED 6F		
RRD								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F

ジャンプ，コール，リターン

×	UN COND	C	NC	Z	NZ	PE	PO	M	P	
JP ×, nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JP ×, e	18 e−2	38 e−2	30 e−2	28 e−2	20 e−2					
JP (HL)	E9									
JP (IX)	DD E9									
JP (IY)	FD E9									
CALL ×, nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DJNZ e										10 e−2
RET ×	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETI	ED 4D									
RETN	ED 45									

リスタート

RST 00H	C7
RST 08H	CF
RST 10H	D7
RST 18H	DF
RST 20H	E7
RST 28H	EF
RST 30H	F7
RST 38H	FF

ビット操作

×	A	B	C	D	E	H	L	(HL)	(IX +d)	(IY +d)
BIT 0, ×	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
BIT 1, ×	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
BIT 2, ×	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
BIT 3, ×	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
BIT 4, ×	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
BIT 5, ×	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
BIT 6, ×	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
BIT 7, ×	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RES 0, ×	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
RES 1, ×	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
RES 2, ×	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
RES 3, ×	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
RES 4, ×	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
RES 5, ×	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
RES 6, ×	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
RES 7, ×	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET 0, ×	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
SET 1, ×	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
SET 2, ×	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
SET 3, ×	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
SET 4, ×	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
SET 5, ×	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
SET 6, ×	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
SET 7, ×	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

入　力

IN A, n	DB n
IN A, (C)	ED 78
IN B, (C)	ED 40
IN C, (C)	ED 48
IN D, (C)	ED 50
IN E, (C)	ED 58
IN H, (C)	ED 60
IN L, (C)	ED 68
INI	ED A2
INIR	ED B2
IND	ED AA
INDR	ED BA

出　力

OUT n, A	D3 n
OUT (C), A	ED 79
OUT (C), B	ED 41
OUT (C), C	ED 49
OUT (C), D	ED 51
OUT (C), E	E 59
OUT (C), H	ED 61
OUT (C), L	ED 69
OUTI	ED A3
OTIR	ED B3
OUTD	ED AB
OTDR	ED BB

Z80・インテル8080ニモニック対照表

［注1］表で使われている記号
r ………8ビット・レジスタ（A, B, C, D, E, H, L）
n ………8ビットの数値（00 - FF）
nn………16ビットの数値（0000 - FFFF）
e ………相対アドレッシングの変位置
［注2］相対ジャンプ命令はインテル8080には本来ありませんが、PC - 8801のモニタでは使用できるようになっているため掲載してあります。

命令群＼CPU	ニモニック			
	Z80		インテル8080	
8ビット・ロード命令	LD	r, r	MOV	r, r
	LD	r, n	MVI	r, n
	LD	r, (HL)	MOV	r, M
	LD	(HL), r	MOV	M, r
	LD	(HL), n	MVI	M, n
	LD	A, (BC)	LDAX	B
	LD	A, (DE)	LDAX	D
	LD	A, (nn)	LDA	nn
	LD	(BC), A	STAX	B
	LD	(DE), A	STAX	D
	LD	(nn), A	STA	nn
16ビット・ロード命令	LD	BC, nn	LXI	B, nn
	LD	DE, nn	LXI	D, nn
	LD	HL, nn	LXI	H, nn
	LD	SP, nn	LXI	SP, nn
	LD	HL, (nn)	LHLD	nn

命令群	Z 8 0		インテル8080	
16ビット・ロード命令	LD	(nn), HL	SHLD	nn
	LD	SP, HL	SPHL	
	PUSH	AF	PUSH	PSW
	PUSH	BC	PUSH	B
	PUSH	DE	PUSH	D
	PUSH	HL	PUSH	H
	POP	AF	POP	PSW
	POP	BC	POP	B
	POP	DE	POP	D
	POP	HL	POP	H
エクスチェンジ命令	EX	DE, HL	XCHG	
	EX	(SP), HL	XTHL	
8ビット算術論理演算命令	ADD	A, r	ADD	r
	ADD	A, n	ADI	n
	ADD	A, (HL)	ADD	M
	ADC	A, r	ADC	r
	ADC	A, n	ACI	n
	ADC	A, (HL)	ADC	M
	SUB	r	SUB	r
	SUB	n	SUI	n
	SUB	(HL)	SUB	M
	SBC	A, r	SBB	r
	SBC	A, n	SBI	n

命令群	Z 8 0		インテル8080	
8ビット算術論理演算命令	SBC	A, (HL)	SBB	M
	AND	r	ANA	r
	AND	n	ANI	n
	AND	(HL)	ANA	M
	OR	r	ORA	r
	OR	n	ORI	n
	OR	(HL)	ORA	M
	XOR	r	XRA	r
	XOR	n	XRI	n
	XOR	(HL)	XRA	M
	CP	r	CMP	r
	CP	n	CPI	n
	CP	(HL)	CMP	M
	INC	r	INR	r
	INC	(HL)	INR	M
	DEC	r	DCR	r
	DEC	(HL)	DCR	M
16ビット算術演算命令	ADD	HL, BC	DAD	B
	ADD	HL, DE	DAD	D
	ADD	HL, HL	DAD	H
	ADD	HL, SP	DAD	SP
	INC	BC	INX	B
	INC	DE	INX	D

命令群	Z 8 0		インテル8080	
16ビット算術演算命令	INC	HL	INX	H
	INC	SP	INX	SP
	DEC	BC	DCX	B
	DEC	DE	DCX	D
	DEC	HL	DCX	H
	DEC	SP	DCX	SP
アキュムレータ操作命令	DAA		DAA	
	CPL		CMA	
	CCF		CMC	
	SCF		STC	
CPUコントロール命令	NOP		NOP	
	HALT		HLT	
	DI		DI	
	EI		EI	
ローテート・シフト命令	RLCA		RLC	
	RLA		RAL	
	RRCA		RRC	
	RRA		RAR	
ジャンプ・コール・リターン命令	JP	nn	JMP	nn
	JP	NZ, nn	JNZ	nn
	JP	Z, nn	JZ	nn
	JP	NC, nn	JNC	nn
	JP	C, nn	JC	nn

命令群	Z 8 0		インテル8080	
ジャンプ・コール・リターン命令	JP	PO, nn	JPO	nn
	JP	PE, nn	JPE	nn
	JP	P, nn	JP	nn
	JP	M, nn	JM	nn
	JR	e	JMPR	e
	JR	NZ, e	JRNZ	e
	JR	Z, e	JRZ	e
	JR	NC, e	JRNC	e
	JR	C, e	JRC	e
	JP	(HL)	PCHL	
	DJNZ	e	DJNZ	e
	CALL	nn	CALL	nn
	CALL	NZ, nn	CNZ	nn
	CALL	Z, nn	CZ	nn
	CALL	NC, nn	CNC	nn
	CALL	C, nn	CC	nn
	CALL	PO, nn	CPO	nn
	CALL	PE, nn	CPE	nn
	CALL	P, nn	CP	nn
	CALL	M, nn	CM	nn
	RET		RET	
	RET	NZ	RNZ	
	RET	Z	RZ	

命令群	Z 8 0	インテル8080
ジャンプ・コール・リターン命令	RET NC	RNC
	RET C	RC
	RET PO	RPO
	RET PE	RPE
	RET P	RP
	RET M	RM
リスタート命令	RST 0H	RST 0
	RST 8H	RST 1
	RST 10H	RST 2
	RST 18H	RST 3
	RST 20H	RST 4
	RST 28H	RST 5
	RST 30H	RST 6
	RST 38H	RST 7
入出力命令	In A, (n)	IN n
	OUT (n), A	OUT n

〈著者紹介〉

日高　徹（ひだか　とおる）

1949年　栃木県宇都宮市生まれ。
早稲田大学商学部卒業後、商社やカー用品メーカーに十数年勤務。現在はフリーのゲームデザイナー。
市販された作品に『ホーンテッドケイブ』『マジックガーデン』『北斗の拳』『ガンダーラ』など。著書に『マシン語ゲームプログラミング』『マシン語サウンド・プログラミング』『マシン語ゲームグラフィックス』（以上、アスキー）、『PC-9800シリーズ はじめてのマシン語』『Z80マシン語秘伝の書』（以上、啓学出版）がある。
趣味はトレーニング。剣道三段。運転免許証を全種類持つ隠れプロドライバーでもある。

PC-8800シリーズ
はじめてのマシン語

1987年7月1日　第1刷発行
1990年11月30日　第6刷発行

著　者　日高　徹（ひだか　とおる）

発行所　啓学出版株式会社
代表者　三井数美
郵便番号　101
東京都千代田区神田神保町1-46
電　話　東京03(233)3731(代)
振　替　東京3-109286

印刷／昭和工業写真印刷所
製本／徳往製本所
本書の定価はカバーに表示してあります

ISBN4-7665-0274-4

Printed in Japan　　TOK

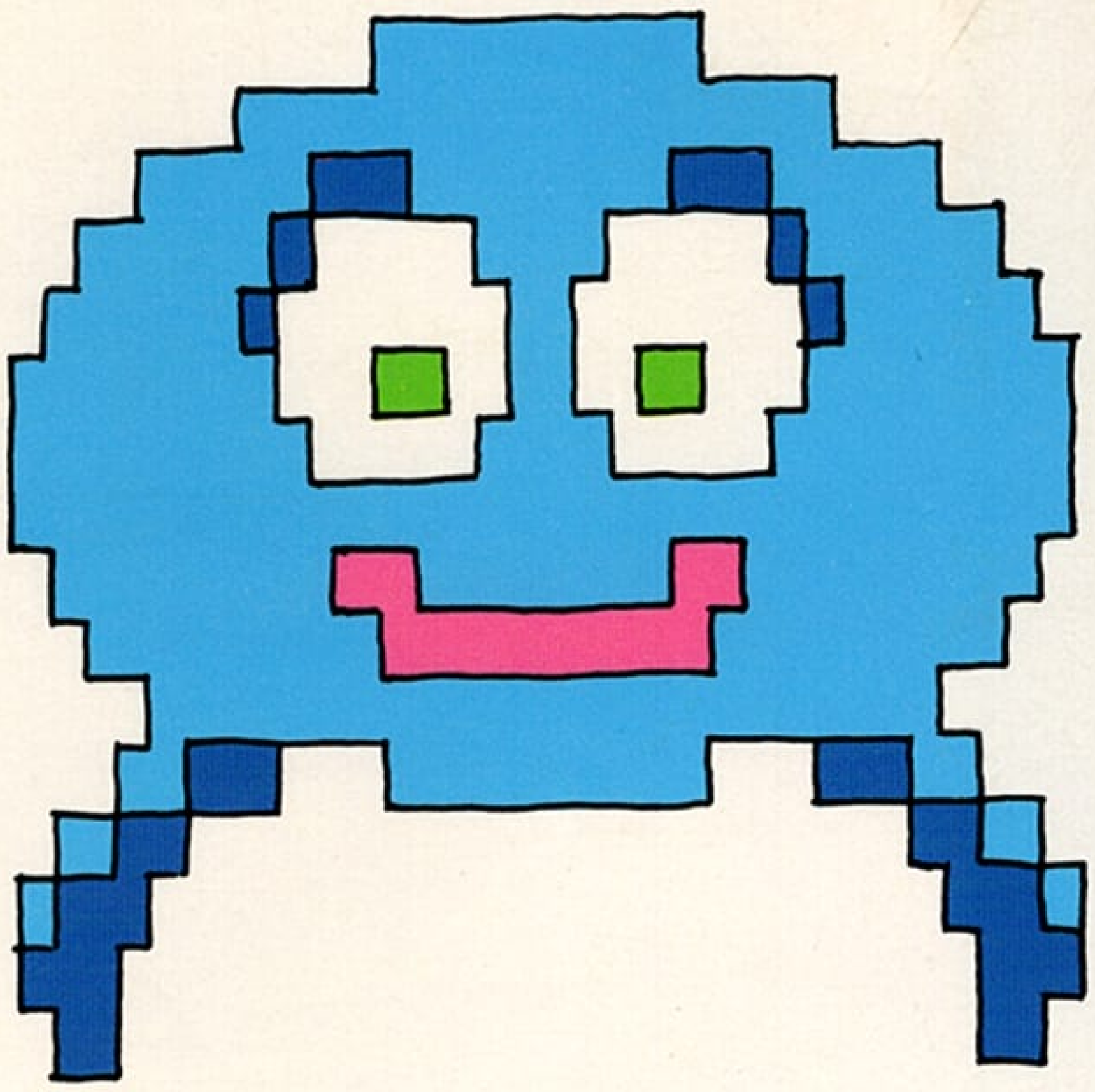
マシン語の基本について知りたい」との感想がわりと多く寄せられたからなんです。
その手紙を読んだとき，わたしは，むかしの自分自身と同じ姿を思い浮かべ，いまの自分なら分かってしまっていることでも，はじめての人にはそうでないんだということにあらためて思い至ったわけですよ。初心を忘れていたなァって。
高いお金を出してアセンブラを買ってマシン語プログラミングを本格的に始める前に，この本に一度目を通してもらうといいと思います。
PC-8801というパソコンは，モニタのアセンブラを内蔵しているという点で，マシン語の基礎を学ぶには絶好の機種なんですね。そのメリットを大いに活用しましょうよ。

●**著者紹介**

日高　徹（ひだか　とおる）
1949年　栃木県宇都宮市生まれ。
早稲田大学商学部卒業後，商社やカー用品メーカーに十数年勤務。現在はフリーのゲームデザイナー。
市販された作品に『ホーンテッドケイブ』『マジックガーデン』『北斗の拳』『ガンダーラ』など。
著書に『マシン語ゲームプログラミング』がある。
趣味はトレーニング。剣道三段。運転免許証を全種類持つ隠れプロドライバーでもある。

PC-8800シリーズ
はじめてのマシン語
日高徹・著

[出] 1953
啓学出版
定価1960円（本体1903円）



ISBN4-7665-0274-4 C3055 P1960E